Software Productivity Analysis and Cost Estimation

The 2nd International Workshop, SPACE 2008
2 December 2008
Beijing, China

Chair/Editor:
Jacky Keung
(NICTA, Australia)

APSEC 2008 Workshop Proceedings
Conference Chairs

General Chair

Dr. Jacky Keung
Empirical Software Engineering
National ICT Australia Ltd. (NICTA)
Sydney, Australia

Research Program Chair

Prof. Ross Jeffery
Empirical Software Engineering
National ICT Australia Ltd. (NICTA)
Sydney, Australia

Publicity Chair

Assoc. Prof. Makoto Nonaka
Faculty of Business Administration
Toyo University
Tokyo, Japan
Program Committee

Prof. Qing Wang
Prof. Barbara Kitchenham
Prof. Ross Jeffery
Dr. Jacky Keung
Dr. Liam O’Brien
Dr. Mahmood Niazi
Dr. Sarah Beecha
Prof. Magne Jorgensen
Dr. Jurgen Munch
Dr. JingYue Li
Prof. Stephen MacDonell
Prof. Hajimu Iida
Mr. Yasutaka Kamei
Prof. Hironori Washizaki
Dr. Makoto Nonaka
Dr. Naoki Ohsugi
Prof. Hirohisa Aman
Ms. Ana Magazinovic
Prof. Deron Liang
Prof. Beyongju Choi
Prof. Ho-Jin Choi

ISCAS, China
Keele University, UK
NICTA, Australia
NICTA, Australia
NICTA, Australia
Keele University, UK
Hertfordshire University, UK
SIMULA Lab, Norway
IESE, Germany
NTNU, Norway
AUT, New Zealand
NAIST, Japan
NAIST, Japan
NII, Japan
Toyo University, Japan
NTT Data, Japan
Ehime Uni, Japan
Chalmers Uni, Sweden
NTO Uni, Taiwan
Ewha Women’s Uni., Korea
Info. & Comm. Uni., Korea
Welcome to the First International Workshop on Software Productivity Analysis and Cost Estimation (SPACE 2008), held in conjunction with the IEEE Asia-Pacific Conference on Software Engineering (APSEC 2008), Beijing, China.

Software project managers require reliable methods for estimating software project costs, and assessing software development productivity. For over 25 years, there has been considerable research effort directed towards software cost estimation and software productivity analysis, various algorithmic approaches developed and their performance reported in the research literature. But nevertheless, cost estimation and productivity analysis remains a complex problem in the software industry.

The goal of the workshop on software productivity analysis and cost estimation, SPACE 2008, is to bring together practitioners and researchers for discussion and presentation on the emerging aspects pertaining to software cost estimation, productivity analysis, prediction models and techniques, and lessons learned. The workshop provides a leading forum to present new ideas and to explore future directions in these areas for software engineering and software project management.

I would like to take this opportunity to thank the following, who have all made significant contributions to the success of SPACE 2008:

- The APSEC 2008 organizing committee
- The SPACE 2008 organizing committee
- Members of the program committee
- The keynote speaker Dr. Stefan Wagner

And finally, thanks to all the participants in SPACE 2008, especially those of you from overseas. We hope you find the workshop intellectually stimulating, as well as enjoying some of the many attractions that the old Beijing city has to offer.

Dr. Jacky Keung
General Chair, SPACE 2008
NICTA (Sydney, Australia)
Biography

Dr. Jacky Keung is a Research Scientist in the Empirical Software Engineering Research Group at National ICT Australia (NICTA), based in Sydney. NICTA is Australia’s centre of excellence for Information and Communications Technology R&D. He also holds an academic position in the School of Computer Science and Engineering at the University of New South Wales, Sydney, Australia. He completed his B.S (Hons) in Computer Science from the University of Sydney, and received his Ph.D from the University of New South Wales for his research into statistical methods of software cost estimation. Jacky works for NICTA in a range of technical roles including consulting in software measurement and cost estimation for a number of software engineering organizations in Australia and Asia. His current research interests are in software measurement and its application to project management, cost estimation, quality control and risk management, as well as software process improvement. He is a member of the Australian Computer Society, and a member of the IEEE Computer Society.
# Table of Contents

✦ **Keynote**

The Chimera of Software Productivity: Effectiveness and Efficiency in Software Evolution..........................1  
*Dr. Stefan Wagner (Technical University of Munich, Germany)*

✦ **Accepted Research Papers**

1. **Empirical Evaluation of Missing Data Techniques for Effort Estimation**.................................3  
   *Koichi Tamura, Takeshi Kakimoto, Koji Toda, Masateru Tsunoda, Akito Monden, Ken-ichi Matsumoto*

2. **A Systematic Review of Productivity Factors in Software Development**..............................11  
   *Stefan Wagner, Melanie Ruhe*

3. **Comparative Study on Prediction Intervals of Effort Estimation Models Based on Linear Regression Models**.................................................................19  
   *Sousuke Amasaki*

4. **Using a Bayesian Network in the ProdFLOW Approach**.........................................................27  
   *Stefan Wagner, Melanie Ruhe*

5. **Lessons Learned in Evaluating Software Engineering Prediction Systems**..........................35  
   *Wasif Afzal, Richard Torkar*

6. **Software Error Impact and Cost Analysis**...............................................................................45  
   *Ralf Gitzel, Simone Krug, Martin Schader*

7. **Effect of Functional Similarity for Establishing Relation between Effort and Functional Size**  
   *Seckin Tunalilar, Onur Demirors*

8. **Quantified Runtime Performance Analysis of Java Patterns**................................................61  
   *Dapeng Liu, Shaochun Xu, Monica Brockmeyer*
The Chimera of Software Productivity: Effectiveness and Efficiency in Software Evolution

Software evolution is the term used in software engineering to refer to the process of developing software initially, then repeatedly updating it. Clearly, it is an essential goal to minimize the cost and to maximize the benefits of software evolution. With the present increasing dependency on large scale software systems, the ability to develop and change existing software in a timely and economical manner is essential for most enterprises. The term associated with this ability is "productivity" that traditionally refers to the ratio between the quantity of software produced and the cost spent for it. The software engineering community has so far been unable to develop a thorough understanding of productivity in software evolution and the significance of the factors influencing it, let alone universally valid methods and tools to analyze, measure, compare and improve productivity. What complicates the situation is the lack of an established, clearly defined terminology. We discuss the terms frequently associated with productivity, namely efficiency, effectiveness, performance, and profitability as a first step towards a more mature management of productivity. Then initial results from the experience with a descriptive, activity-based model of productivity are presented. That model captures the important factors of software productivity and serves as basis for further productivity analyses that have been applied in practice.

Biography

Dr. Stefan Wagner holds a diploma degree from the University of Applied Sciences Augsburg, an MSc from Heriot-Watt University, Edinburgh, and a doctoral degree from Technische Universität München. He is currently a post-doctoral researcher at TU München and works in the areas of software quality assurance and management as well as software economics and productivity. He has experiences from several industrial and academic projects on these topics. He is a member of the IEEE Computer Society, ACM SIGSOFT, and the Gesellschaft für Informatik.
Research Papers
Empirical Evaluation of Missing Data Techniques for Effort Estimation

Koichi Tamura, Takeshi Kakimoto, Koji Toda, Masateru Tsunoda, Akito Monden, Ken-ichi Matsumoto
Empirical Evaluation of Missing Data Techniques for Effort Estimation

Koichi Tamura¹, Takeshi Kakimoto², Koji Toda¹, Masateru Tsunoda¹, Akito Monden¹, Ken-ichi Matsumoto¹

¹ Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
{koichi-t, koji-to, masate-t, akito-m, matumoto}@is.naist.jp

² Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka, 560-8531, Japan
kakimoto@ist.osaka-u.ac.jp

Abstract

Multivariate regression models have been commonly used to estimate the software development effort to assist project planning and/or management. These models require a complete data set that has no missing values for model construction. The complete data set is usually built either by using imputation methods or by deleting projects and/or metrics that have missing values (we call this RC deletion). However, it is unclear which method is the most suitable for the effort estimation. In this paper, using the ISBSG data set of 706 projects (containing 47% missing values) collected from several companies, we applied four imputation methods (mean imputation, pairwise deletion, k-NN method and CF method) and RC deletion to build regression models. Then, using a data set of 143 projects (with no missing values), we evaluated the estimation performance of models after applying each imputation or the RC deletion. The result showed that the similarity-based imputation method (k-NN method and CF method) showed better performance than other methods in terms of MdMAE, MdMRE, MdMER and Pred(25).

1. Introduction

In a software development project, software cost estimation is necessary for management of schedule and resources. So far, various quantitative estimation methods using a historical project data set have been proposed and used [3] [12] [14]. Among these methods, the regression model has been most widely used for its convenience [1] [9] [15].

One of the practical problems in using estimation methods is that the historical project data usually contain substantial numbers of missing values [1] [11]. One reason is that different divisions in an organization might have different policies on data collection, i.e. one project collects a particular metric while other projects do not. Even if the organization has a unified policy, not all metrics are collected in each project due to the pressing development schedule. However, not only regression models but also many other estimation models need a data set with no missing values to build models.

One approach to solve this problem is to delete the metrics and projects with missing values from the data set (we call this RC deletion). This method is easy to use; however, deletion might remove useful information for effort estimation. In addition, there are unclear trade-offs between deletion of metrics and deletion of projects. Another commonly used approach is using imputation methods [6] [8] [11]. Imputation does not reduce the information; however, it might introduce noise to the data set. Yet another approach is pairwise deletion, which is applicable to regression models [15]. While all these methods are considered useful, it is unclear which is the best. It is important for an engineer to know which is better: (a) to avoid losing information despite introducing noise or (b) to avoid noise despite losing information.

Our primary goal is to clarify which missing data technique shows the best performance for building the regression model. In this paper, using a data set of 706 projects (containing 47% missing values) collected by the International Software Benchmarking Standards Group (ISBSG) [5], we applied four imputation methods (mean imputation, pairwise deletion, k-NN method and CF method) and RC deletion to build estimation models using stepwise multiple regression analysis [2]. Then, using a data set of 143 projects (with no missing values), also from the ISBSG data set, we evaluated the estimation
performance of models after applying each imputation or the RC deletion. The CF method is based on collaborative filtering and applying it to missing data imputation is a first-time approach.

2. Imputation and deletion methods

In this paper, the following methods were applied to the data set before building estimation models.

2.1. Mean imputation

This method fills each missing value with the mean of observed values [8] [11] [15].

2.2. Pairwise deletion

This method is particularly associated with the regression analysis [8] [15]. While calculating a correlation matrix to build a regression model, a correlation between each pair of variables is calculated from all cases (projects) that have non-missing value on those two variables. This method is widely used in statistical analysis tools such as SPSS.

2.3. Similarity-based imputation (k-NN method)

This method uses existing metrics values of k most similar projects to fill missing values of the target project. The degree of similarity between two projects is computed by Euclidean distance [1] [6].

2.4. Similarity-based imputation (CF method)

This method is an alternative similarity-based imputation based on collaborative filtering [12]. Although this method is used for cost estimation, we newly apply it to missing data imputation. The method uses cosine similarity to compute the similarity instead of Euclidean distance. The three-step procedure of the CF method is described below.

Step 1 (normalization of metrics): Since each metric has a different value range, this first step normalizes values of metrics so that the value range becomes [0, 1]. Here, we denote that $p_i$ is $i$-th project, $m_j$ is $j$-th metric, and $v_{ij}$ is the value of metric $m_j$ observed in project $p_i$. The normalized value $v'_{ij}$ of $v_{ij}$ (of project $p_i$) is calculated by the following equation:

$$v'_{ij} = \frac{v_{ij} - \min(P_j)}{\max(P_j) - \min(P_j)}$$

(1)

where $P_j$ denotes a set of projects in which the value of metric $m_j$ was observed (collected), $\max(P_j)$ and $\min(P_j)$ denote the maximum and minimum value in $\{v_{ij} \mid p_i \in P_j \}$ respectively.

Step 2 (computation of similarity between projects): In this step, similarity $sim(p_o, p_i)$ between the target project $p_o$ and other projects $p_i$ is computed. Formally, we can define the $sim(p_o, p_i)$ between the target project $p_o$ and other projects $p_i$ as:

$$sim(p_o, p_i) = \frac{\sum_{j \in M_o, m_j} (v'_{o,j} - \text{md}(m'_j)) \times (v'_{i,j} - \text{md}(m'_j))}{\sqrt{\sum_{j \in M_o, m_j} (v'_{o,j} - \text{md}(m'_j))^2 \times \sum_{j \in M_o, m_j} (v'_{i,j} - \text{md}(m'_j))^2}}$$

(2)

where $M_o$ and $M_i$ denote a set of metrics observed in project $p_o$ and $p_i$ respectively, $m_j$ denotes the normalized value of $m_j$, and $\text{md}(m'_j)$ denotes the median of $m'_j$.

The metrics that are higher than $\text{md}(m'_j)$ show positive values and the metrics that are lower than $\text{md}(m'_j)$ show negative values by subtracting $\text{md}(m'_j)$. The value range of $sim(p_o, p_i)$ is [-1, 1] for this computation. Note that $sim(p_o, p_i)$ shows low value (i.e., the computed similarity shows low value) if the difference of metrics between $p_o$ and $p_i$ is great.

Step 3 (computation of estimation): This step calculates an estimated value $\hat{v}_{a,b}$ of the metric $m_i$ on the target project $p_a$ using $sim(p_o, p_i)$ calculated in the previous step. The estimated value is computed as the sum of the metrics’ values given by the other projects similar to $p_o$. Each value is weighted by the corresponding $\text{amplifier}(p_o, p_i)$ and $sim(p_o, p_i)$ between $p_a$ and $p_i$. Formally, we can define the estimated value as:

$$\hat{v}_{a,b} = \frac{\sum_{i \in k-nearestProjects} (v_{i,b} \times \text{amplifier}(p_o, p_i) \times sim(p_o, p_i))}{\sum_{i \in k-nearestProjects} \sum_{i \in k-nearestProjects} sim(p_o, p_i)}$$

(3)

where $k$-nearestProjects denotes a set of $k$ projects (called neighborhoods) that have the highest similarity with $p_o$. The neighborhoods must have $m_i$ as an observed metric. Generally, the neighborhood size $k$ affects the estimation accuracy (This point applies to the k-NN method as well).

To improve accuracy of the estimation, the $\text{amplifier}(p_o, p_i)$ calculates an approximate value of the $v_{i,b}$ with comparing the sizes of projects $p_a$ and $p_i$, i.e. the amplifier indicates what times $p_a$’s value is $p_i$’s value. The $\text{amplifier}$ derived from the fact that the $p_a$’s value is several times larger (or smaller) than the $p_i$’s value when $p_i$ is similar to $p_a$. It’s because the similarity is computed by vector operation but not Euclidean distance. $sim(p_o, p_i)$ is computed by comparing tendencies of the values, whereas Euclidean distance is computed by comparing
absolute values. Formally, we can define the \( \text{amplifier}(p_a, p_i) \) as:

\[
\text{amplifier}(p_a, p_i) = \frac{f_a}{f_i} \tag{4}
\]

where \( f_a \) denotes the function points of project \( p_a \).

2.5. Row-column deletion method (RC deletion)

This method deletes the projects and/or metrics with missing values from a data set to build a complete data set. Listwise deletion \([8]\) \([15]\) is a subset of this method, which only deletes projects with missing values. The RC deletion allows deleting metrics to reduce the projects to be deleted. There exists a trade-off between deletion of metrics and deletion of projects.

3. Experiment

3.1. Overview

Using the ISBSG data set, we experimentally compare missing data techniques (imputation methods and the RC deletion) by evaluating the prediction performance of effort estimation models after applying the techniques.

We cannot evaluate the accuracy of imputed (filled) values by comparing true values with imputed values since we use the data set that originally contained missing values. Therefore, using a test data set with no missing values, we indirectly evaluate the performance of imputation methods by evaluating the performance of effort estimation for the test data.

3.2. Dataset

In the experiment, we used the ISBSG data set, collected from 20 nations’ software development companies \([5]\) and with many missing values (missing value ratio was 58\%). Furthermore, the ISBSG data set has been widely used for past empirical experimentations that evaluated various missing data techniques and estimation methods \([9]\) \([10]\) \([13]\).

We assumed that to estimate the effort is the end of the design phase. The extracted data set of 849 projects (missing value ratio was 39\%) whose summary work effort (estimation target) was recorded (i.e. not missing), the count approach of FP was <IFPUG>, the development type was <new development> and the data quality rating was <A> or <B> \([9]\).

Table 1 presents metrics contained in the ISBSG data set. We chose the four metrics to estimate the objective variable. Although <Project elapsed time> is an actually measured value, we included it as a predictor variable since (1) it probably affects productivity and cost, (2) the planned value of the project elapsed time is not collected, (3) the project duration is usually fixed in the initial stage of a project and <Project elapsed time> is usually not widely different from its planned value. Mean imputation and similarity-based imputation focus on filling the numerical variable, and then we removed categorical variables such as development platform, language type and business area type. Furthermore, the number of projects to build regression models falls if we include too many metrics in predictor variables. For these reasons, we chose the four metrics as predictor variables.

We divided the data set according to missing values into the fit data set of 706 projects (containing 47\% missing values) and the test data set of 143 projects (with no missing value). The fit data set is used for building estimation models and the test data set is for evaluation of the estimation performance of built models.

The fit data set has at least one missing value in all projects, so RC deletion first deletes the metrics with many missing values and then deletes the projects with missing values in the fit data set. According to the metrics to be deleted, we built three data sets: by deleting <Effort plan> (72 projects), by deleting <Effort specify> (28 projects) and by deleting both <Effort plan> and <Effort specify> (631 projects). We used \( k = 3 \) in the \( k\text{-NN} \) method and \( k = 8 \) in the CF method whose residual mean square showed the minimum when we built regression models.

3.3. Evaluation criteria

We used four evaluation criteria: magnitude of absolute error (MAE), magnitude of relative error (MRE), magnitude of error relative (MER) \([3]\), and Pred(25) \([14]\). MRE and MER are defined as \((5)\) and \((6)\) respectively as follows (where \( X = \) actual effort, \( \hat{X} = \) predicted effort):

\[
\text{MRE} = \frac{|\hat{X} - X|}{X} \tag{5}
\]

\[
\text{MER} = \frac{|\hat{X} - X|}{\hat{X}}. \tag{6}
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Missing Value Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function points</td>
<td>0%</td>
</tr>
<tr>
<td>Project elapsed time [month]</td>
<td>8.8%</td>
</tr>
<tr>
<td>Effort plan [person-hours]</td>
<td>77.0%</td>
</tr>
<tr>
<td>Effort specify [person-hours]</td>
<td>71.2%</td>
</tr>
<tr>
<td>Objective variable</td>
<td>Summary work effort [person-hours]</td>
</tr>
</tbody>
</table>
MRE and MER are criteria to evaluate overestimation and underestimation respectively [3]. Using either of them is insufficient because even if the MRE of a model is small, the model might overestimate if MER is much greater than MRE. Pred(25) is the percentage of predictions that fall within 25 percent of the MRE.

3.4. Experimental procedure

The experimental procedure is as follows.

Step 1. We apply each missing data technique to the fit data set.
Step 2. Using the fit data set, we apply a stepwise regression analysis to build an estimation model whose objective variable is <Summary work effort>.
Step 3. Considering that the summary work effort of the test data set is unknown, we estimate the summary work effort of the test data set with built models and calculate the value of each evaluation criterion.

4. Results and discussion

4.1. Overall results

Table 2 shows the median of MAE, MRE and MER (MdMAE, MdMRE and MdMER), and Pred(25) of estimations when we applied each missing data technique. Fig. 1 and Fig. 2 show box plots of the MRE and MER values when each technique is applied respectively. In Table 2, Fig. 1 and Fig. 2, “P deletion” stands for a data set built by deleting Effort plan, and “S deletion” for Effort specify. The result showed that the similarity-based imputation methods (k-NN method and CF method) showed better performance than other methods in all criteria.

We ascertain whether each evaluation criterion is statistically significant. The value of MAE, MRE, MER is not normally distributed, so we use the non-parametric Wilcoxon matched-pairs signed-ranks test to assess their statistical differences. And we use a chi-square test for Pred(25). The level of significance is 0.05 in each test. The result of each test is described in the following section.

4.2. Comparison of similarity-based imputation and other imputation

Among three imputation methods, similarity-based imputation methods (k-NN method and CF method) were much better than the mean imputation (see Table 2). This result follows the past research using artificial missing values [15]. Similarity-based methods were also much better than the pairwise deletion. As shown in Fig. 1 and Fig. 2, inter-quartile range (IQR) of similarity-based imputation methods was narrower than mean imputation and pairwise deletion, i.e. the variability of the errors is lower. Furthermore, each test of the result of similarity-based imputation and the other imputation showed that each evaluation criterion was statistically significant.

4.3. Comparison of similarity-based imputation and RC deletion

As shown in Table 2, similarity-based imputation methods were also better than all RC deletions (P deletion, S deletion and P, S deletion). In the RC deletions, although there was no significant difference between P deletion and S deletion in MdMRE and Pred(25), P deletion was better than S deletion in MdMAE and MdMER. Although 631 projects remained in the P, S deletion data set, it seems deleting both P (Effort plan) and S (Effort specify) removed useful information for the estimation. On the other hand, it can be considered that the P deletion data set (72 projects) and the S deletion data set (28 projects) contained too few projects to build an accurate model, i.e. the confidence interval might become wide.

As shown in Fig. 1 and Fig. 2, IQR of the similarity-based imputation method (k-NN method) was narrower than three RC deletions. IQR of the CF method was narrower than three RC deletions in MRE. Although IQR of the CF method was wider than P deletion, the position of the boxplot of the CF method was lower than three RC deletions. Therefore, the CF method was better than three RC deletions. Furthermore, each test of the result of the similarity-based imputation and RC deletion showed that...
each evaluation criterion was statistically significant except for MER between the CF method and RC deletion (P deletion).

This indicates that the large project data set with a lot of missing values is worthier than the small project data set with no missing value. We believe engineers should not be afraid of usage of “incomplete” project data since it is useful enough for the effort estimation. In addition, interestingly, all RC deletions still showed better performance than the mean imputation. This indicates that the incomplete data set is useful only if missing values are properly filled in.

4.4. Comparison of similarity-based imputation methods

Comparing two similarity-based imputation methods, the $k$-NN method was better than the CF method in terms of MdMAE and MdMER, while this became the opposite in terms of MdMRE and Pred(25) (see Table 2). As shown in Fig. 1 and Fig. 2, IQR of the $k$-NN method was narrower than the CF method in MRE and MER. Furthermore, each test of the result of the CF method and $k$-NN method showed that MAE and MER were statistically significant.

From these results, there is not a big difference between the $k$-NN method and CF method in their prediction performance. However, we could say the $k$-NN method tends to build an overestimate model, while the CF method tends to build an underestimate model. Therefore, it is preferable for an estimator to use both methods to confirm that there is no big difference in their estimates.

4.5. Threats to validity

We evaluated using only one data set; however, there are many conditions (i.e., various amounts and distributions of the missing values) in the data set collected by the actual industrial organizations. Other experimental simulation would have to be performed on a different data set for improving reliability of the study.

Our experimentation focused on stepwise multiple regression analysis as the modeling technique. This is one of the most popular techniques, such as ordinary least squares regression analysis, for building cost estimation models. However, it is possible that other modeling techniques would require different missing data techniques.

5. Related work

Jonsson et al. [6] focused on filling in missing data accurately. They artificially deleted some recorded values at random and compared filled in values with originally recorded values. The result showed that the $k$-NN method using incomplete case strategy and $k$-value of the square root of complete cases performed well.

Sentas et al. [13] and Strike et al. [15] compared several imputation methods and listwise deletion, but they used a data set whose values were artificially deleted at random. In [13], multinomial logistic regression showed better performance than listwise deletion, mean imputation, expectation maximization and regression imputation for estimating categorical missing values. In [15], they have compared listwise deletion, mean imputation and eight different types of hot-deck imputation. The results showed that the $k$-NN method (Euclidean distance) and a z-score standardization showed the best performance.

Although the data set whose values were deleted at random was used in the experiment ([6] [13] [15]), the missing data pattern is realistically not random but burst. There are several reasons of the burst missing. The burst missing is generated if several data sets collected by different business organizations having different policies on data collection are merged. Even with the same organization, the burst missing is generated if its policy is different from each period.
Cartwright et al. [1] and Myrtveit et al. [11] compared imputation methods by evaluating the goodness of fit of effort estimation models built after imputations were applied to a fit data set containing missing values. Therefore, they did not evaluate the prediction performance of the models. But it remains possible that the built models too much fit (i.e., overfitting [4]) the fit data set, and then the models might not perform well when they evaluate the estimation accuracy to another data set.

In this paper, we have done imputation or RC deletion to a data set containing missing values naturally and prepared a test data set with no missing values to do this evaluation. Furthermore, none of the past research compared imputation methods with RC deletion, which is often used in industry.

6. Conclusion

In this paper, we experimentally compared missing data techniques (mean imputation, pairwise deletion, k-NN method, CF method and RC deletion) by evaluating the prediction performance of effort estimation models after applying the techniques. Our findings include:

- Similarity-based imputation methods (k-NN method and CF method) showed better performance than all RC deletions. This indicates that a large project data set with a lot of missing values is worthier than a small data set with no missing value.
- The mean imputation was worse than all three RC deletions. This indicates that an incomplete data set becomes useful only if its missing values are properly filled in.
- The pairwise deletion was worse than similarity-based imputation.
- The k-NN method tends to build an overestimate model, while the CF method tends to build an underestimate model. We recommend an estimator to use both methods to confirm there is no big difference in their estimates.

Our future work will be to use other data sets to increase the validity of the results. Furthermore, we will develop similarity-based imputation method to improve imputation performance and make comparison of the cost estimation performances with other methods given missing data (e.g. optimized set reduction).

7. Acknowledgments

This work is being conducted as a part of the StagE Project, the Development of Next Generation IT Infrastructure, supported by Ministry of Education, Culture, Sports, Science and Technology.

8. References

A Systematic Review of Productivity Factors in Software Development

Stefan Wagner, Melanie Ruhe
A Systematic Review of Productivity Factors in Software Development

Stefan Wagner
Technische Universität München
Garching b. München, Germany
wagnerst@in.tum.de

Melanie Ruhe
Siemens AG
Munich, Germany
melanie.ruhe@siemens.com

Abstract
Analysing and improving productivity has been one of the main goals of software engineering research since its beginnings. A plethora of studies has been conducted on various factors that resulted in several models for analysis and prediction of productivity. However, productivity is still an issue in current software development and not all factors and their relationships are known. This paper reviews the large body of available literature in order to distill a list of the main factors influencing productivity investigated so far. The measure for importance here is the number of articles a factor is mentioned in. Special consideration is given to soft or human-related factors in software engineering that are often not analysed with equal detail as more technical factors. The resulting list can be used to guide further analysis and as basis for building productivity models.

1. Introduction
Productivity in software development has been an important research area for several decades. It is the key for a successful software company to control and improve its productivity. However, in contrast to traditional industrial work, it is hard to measure for software development. There are several terms that are used more or less synonymously such as performance or efficiency. There are also various definitions of which output divided by input is the most general one.

In software development lines-of-code (LOC) and function points (FP) are traditionally used in measures for productivity, i.e., the amount of LOC or FP produced per hour by a developer. Based on this, there is a large amount of studies on various aspects of productivity. The two mentioned measures and several more dimensions have been analysed and detailed. Models have been built that should explain, analyse and predict productivity. Finally, several studies analyse the factors that influence productivity in a software project.

Problem Frese and Brodbeck [12] claim that the scientific discussion about the work situation in software development and about productivity factors in such projects is done based on an insufficient empirical basis. According to them, it is dominated by shallow surveys and qualitative experience reports.

Moreover, the software engineering literature in that area often has a strong emphasis on mainly technical factors such as the software size or the product complexity. However, Brodbeck [7] shows that more than a third of the time a typical software developer is not concerned with technical work. Meetings and talks constitute 21.1%, presentations and project organisation 9.6%, and independent qualification 6% of the work time. Hence, these efforts are significant.

Contribution We provide a systematic review of software engineering, management and organisational psychology literature on productivity factors in software development. A list of these factors is distilled from the literature in order to aid model building, productivity improvement and further research.

The importance of the factors and thereby their inclusion in the list is based on their mentioning in the analysed literature. Hence, it is secured that several authors used and/or analysed the factor. Furthermore, we put specific care to the equal consideration of technical and soft factors in order to represent their significance.

2. Review approach
For the systematic review of the productivity literature, we use a combined approach of automated and manual search. Our aim is to include literature from the areas software engineering, management and organisational psychology as these are the main sources of relevant literature. For this we used a query on four portals for scientific literature that contains the typically used terms in papers about what we call productivity factors. For the automated search we used the following expression:
software AND (productivity OR "development efficiency" OR "development effectiveness" OR "development performance")

It resulted in the following numbers of results:

- ACM’s The Guide: 10,017
- IEEE Xplore: 1,408
- ScienceDirect: 508
- Google Scholar: 680,000

These large numbers show on the one hand the significance of the topic in research but on the other hand prevents the manual analysis of all these papers. We inspected the first 100 results of each portal whether they are suitable for inclusion in our study. In this inspection we omitted very specific analyses of single, detailed factors because of brevity reasons. We also excluded studies that only showed that factors have no influence on productivity. Although these studies are of interest in general, they do not help in building a list of factors that do influence productivity.

In addition to the papers retrieved using that query, we also collected papers manually in a number of important journals in software engineering (e.g. IEEE Transactions on Software Engineering), in management (e.g. Management Science) and organisational psychology (e.g. Journal of Occupational and Organizational Psychology). From these and by following references from the already found papers, the complete body of papers that build the basis of this study was collected. Moreover, the well-known books by Boehm [5] and Jones [18] on software productivity were included as a baseline.

We derived from this body of papers factors about the product, process and people and unified synonymous terms as far as possible. Then the extracted factors were ranked by appearances in the literature. We mainly aimed at finding different authors that used the factors. Based on this, the final list was compiled.

3. Considered studies

Because of space limitation we cannot describe each considered study but we only choose some important representatives of each decade. The full description can be found in [29]. We decided to organise the papers in the order of their publication which has the additional benefit that the developments over time become visible.

3.1. 1970–1979

Walston and Felix [30] analysed in 1977 in one of the first larger studies factors that correlate significantly with programming productivity (measured in effort per SLOC). Several of the later publications use the same or a variant of these factors. A number of the described factors obviously decreased in importance over the decades. For example, chief programmer team usage is not a common practice today. Also with the more and more standardised hardware, previous experience with operational computer does not seem to be a problem anymore. Nevertheless, the majority of factors, such as user participation, overall constraints on program design or previous experience with programming language are still valid.

Albrecht then proposed his famous function points [1]. In this study, he analysed factors like the used programming language and the project size.

3.2. 1980–1989

Brooks [9] uses factors from Walston and Felix [30] as basis in his study at IBM. He found especially the effects of program complexity and structured programming to be important.

Jones started with [17] a series of books about programming productivity. He was one of first that analysed various productivity factors over various domains and could provide industry averages. He focused his measures strongly on LOC and FP.

DeMarco and Lister [11] then aimed in a completely different direction from the LOC- and FP-centred research. They point out that “The major problems of our work are not so much technological as sociological in nature.” They consider turnover as one of the central factors influencing productivity. They also mention the importance of a proper work place with windows, natural light, quietness, etc. They substantiate this by showing that a noisy workplace with a high probability leads to more defects. The used language, years of experience, number of defects and salary do not have an significant effect on productivity in their opinion.

They further claim that “Quality, far beyond that required by the end user, is a means to higher productivity.” They then discuss work interruption as important issue and introduce the E-Factor as ratio of uninterrupted hours and body-present hours as measure for this.

Finally, they list six factors that they called “teamicide”, i.e., measures that are the main obstacles in building (or growing) teams that partially repeat earlier mentioned factors: defensive management, bureaucracy, physical separation, fragmentation of people’s time, quality reduction of the product, phony deadlines, and clique control. In summary, DeMarco and Lister provided in [11] the first and still most comprehensive work on the soft factors influencing productivity in software development.

The most famous model that involves productivity is
COCOMO by Boehm [5]. It is a cost-estimation model in which the productivity of the developers obviously plays a decisive role. These factors have been derived empirically from a large project database. The factors are discussed in more detail in section 3.4 with COCOMO II.

3.3. 1990–1999

The 90s showed, maybe as a result of DeMarco and Lister, a stronger interest in soft factors. Rasch studies in [26] the effect of factors such as team member rotation, role ambiguity and role conflict on job satisfaction and actually quantifies them based on a survey.

Lakanpal [20] concentrated on characteristics of groups and their influence on productivity. The cohesiveness and capability had the strongest influence in 31 development groups, experience had the weakest influence.

Brodbeck describes in [6] that in a survey, the projects with a higher communication effort also were more successful. Even the intensity of internal communication is positively correlated with project success. This is in contrast to common software engineering belief that high communication effort hampers productivity.

Wohlin and Ahlgren describe factors and their impact on time to market in [31]. They use 10 different factors in their study, mostly factors that are covered by the publications discussed so far. They also include product complexity, methods and tools and requirements stability that could be considered technical factors.

Blackburn, Scudder, and Van Wassenhove [3] studied the factors and methods that improved productivity in Western European companies. They found project duration and team size to be significant.

Chatzoglou and Macaulay [10] interviewed participants of over a hundred software projects about several factors and their influence on productivity. They found that experience, knowledge and persistence of the team members is considered important. Also the motivation of the users and their communication with the rest of the team plays a role. Finally, the available resources, tools and techniques used and the management style are important factors.

Glass summarised in [13] his findings on project “runaways”. He states that common causes for such failing projects are that they are huge, that there are usually a multiplicity of causes and that they were aimed to be “breakthroughs” in comparison to older systems. However, he also suggests that technology is increasingly often the cause for project failure.

Hill et al. [15] investigated the influence of virtual offices on aspects of work. Most interestingly in the productivity context is that the perception that “teamwork has been diminished”.

Port and McArthur [24] analysed the introduction of object-oriented methods at Hughes Space and Communications. They found that an object-oriented development approach coupled with object-oriented implementation improves overall project productivity.

3.4. 2000–2007

The most thorough work in the area of productivity and its influencing factors is COCOMO II by Boehm et al. [4]. They have a long experience in that area [5] and derive their factors from a large empirical body. Technical factors they identified are, for example, precedentedness (how similar are the projects) or the product complexity. Boehm et al. also analysed various soft factors and found that those factors combined are more important than all the others. Those factors include programmer capability and personnel continuity (turnover).

Jones states in [18] that software projects are influenced by about 250 factors. Individual projects “are usually affected by ten to 20 major issues.” Of course, he also investigated a series of soft factors. He lists and discusses several factors based on case studies partially with quantitative results. 36 of these factors are considered the major factors. In comparison to other studies, he adds explicitly the support for modern telecommunication facilities such as video conferencing.

Maxwell and Forselius argue in [22] that the influencing factors on productivity depend on the business domain the software is produced for. For example, in the insurance domain requirements volatility, software’s logical complexity and tools use are significant while in the public administration domain number of inquiries and customer participation are of importance.

Kitchenham and Mendes [19] found that reuse is taking place has a significant effect on productivity. The amount of reuse is not that important. They also suggest that the productivity is not significantly different in different countries.

Berntsson-Svensson and Aurum [2] analysed in a survey factors influencing project and product success. They found that different industries define success in different terms. However, the identified influencing factors are similar to other studies: well defined project scope, complete and accurate requirements, good schedule estimations, customer/user involvement, and adding extra personnel.

Mohagheghi and Conradi [23] analysed especially the connection between software reuse and productivity among other factors. They show that there can be a strong positive influence.

Spiegl reports in [28] on a survey on project management issues in software development conducted with project managers. In terms of soft factors he found that support of the top-management, business culture, promotions, team building, relationship management and communica-
Table 1. The derived technical factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>No. of Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedentedness</td>
<td>How similar are the projects?</td>
<td>2</td>
</tr>
<tr>
<td>Required Software Reliability</td>
<td>The level of reliability needed.</td>
<td>3</td>
</tr>
<tr>
<td>Database Size</td>
<td>How large is the data compared to the code?</td>
<td>2</td>
</tr>
<tr>
<td>Product Complexity</td>
<td>The complexity of the function and structure of the software.</td>
<td>6</td>
</tr>
<tr>
<td>Developed for Reusability</td>
<td>To what extent the components should be reusable.</td>
<td>3</td>
</tr>
<tr>
<td>Execution Time Constraints</td>
<td>How much of the available execution time is consumed.</td>
<td>7</td>
</tr>
<tr>
<td>Main Storage Constraint</td>
<td>How much of the available storage is consumed.</td>
<td>3</td>
</tr>
<tr>
<td>Software Size</td>
<td>The amount of code of the system.</td>
<td></td>
</tr>
<tr>
<td>Product Quality</td>
<td>The quality of the product influences motivation and hence productivity.</td>
<td>2</td>
</tr>
<tr>
<td>User Interface</td>
<td>Degree of complexity of the user interface.</td>
<td>3</td>
</tr>
<tr>
<td>Development Flexibility</td>
<td>How strong are the constraints on the system?</td>
<td>2</td>
</tr>
<tr>
<td>Reuse</td>
<td>The extent of reuse.</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>No. of Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Risk Resolution</td>
<td>How are the risks mitigated by architecture?</td>
<td>1</td>
</tr>
<tr>
<td>Process Maturity</td>
<td>The well-definedness of the process.</td>
<td>1</td>
</tr>
<tr>
<td>Platform Volatility</td>
<td>Time span between major changes.</td>
<td>3</td>
</tr>
<tr>
<td>Early Prototyping</td>
<td>Early in the process prototypes are built.</td>
<td>1</td>
</tr>
<tr>
<td>Completeness of Design</td>
<td>The amount of the design that is completed when starting coding</td>
<td>2</td>
</tr>
<tr>
<td>Effective and Efficient V&amp;V</td>
<td>The degree to which defects are found and the needed effort.</td>
<td>1</td>
</tr>
<tr>
<td>Project Duration</td>
<td>Length of the project.</td>
<td>2</td>
</tr>
<tr>
<td>Hardware Concurrent Development</td>
<td>Is the hardware developed concurrently?</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>No. of Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of Software Tools</td>
<td>The degree of tool use.</td>
<td>7</td>
</tr>
<tr>
<td>Programming Language</td>
<td>The level of the used programming language.</td>
<td>3</td>
</tr>
<tr>
<td>Use of Modern Development Practices</td>
<td>Are modern methods applied?</td>
<td>7</td>
</tr>
<tr>
<td>Documentation match to life-cycle needs</td>
<td>How well the documentation fits to the needs.</td>
<td>2</td>
</tr>
</tbody>
</table>


tion, freedom and responsibility, and motivation and appreciation are important.

4. Results

As mentioned above, we roughly divide the productivity factors into technical and soft factors. We see soft factors as all non-technical factors influencing productivity. These factors mainly stem from the team and its work environment. Obviously, the borderline between these two groups is sometimes blurry and is only intended to aid easier comprehension.

The technical factors are summarised in Table 1. In this group we structured the factors in three categories. The product category contains all factors that have a direct relation to the product, i.e., the system itself. The category process is comprised of the technical aspects of the process. Finally, the category development environment contains factors about the tools the developer uses in the project.

The soft factors are summarised in Table 2. Overlapping factors are combined as far as possible. We employed a simple, non-unique categorisation to aid a quick comprehension. Corporate Culture contains the factors that are on a more company-wide level whereas Team Culture denotes similar factors on a team level. In Capabilities and Experiences are factors summarised that are related to individuals.

Environment stands for properties of the working environment. Finally, project-specific factors are in the Project category.

In general, what is surprising in the studies is that communication effort is positive for productivity. It is often discussed that communication should be reduced to decrease “unnecessary” work. However, it seems the problem is only that with increasing people the communication effort increases strongly. Yet, a high fraction of effort on communication seems like a good investment.

Then there is some agreement in the few studies that analysed these factors that the business domain plays a role. Either the domain itself has an influence on productivity or at least it determines which of the other factors have the strongest influence. This contradicts general and generic productivity models but suggests that individual models are needed.

It is also notable that although experience is often brought up and is in interviews considered important, in empirical studies it is rather insignificant. By far more interesting is the capability of the developers. Hence, this suggests that only being in a profession for a long time does not make one productive.
Table 2. The derived soft factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>No. of Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credibility</td>
<td>Open communication and competent organisation.</td>
<td>4</td>
</tr>
<tr>
<td>Respect</td>
<td>Opportunities and responsibilities.</td>
<td>6</td>
</tr>
<tr>
<td>Fairness</td>
<td>Fairness in compensation and diversity.</td>
<td>5</td>
</tr>
<tr>
<td>Corporate Culture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camaraderie</td>
<td>Social and friendly atmosphere in the team.</td>
<td>1</td>
</tr>
<tr>
<td>Team Identity</td>
<td>The common identity of the team members.</td>
<td>2</td>
</tr>
<tr>
<td>Sense of Eliteness</td>
<td>The feeling in the team that they are “superior”</td>
<td>3</td>
</tr>
<tr>
<td>Clear Goals</td>
<td>How clearly defined are the group goals?</td>
<td>3</td>
</tr>
<tr>
<td>Turnover</td>
<td>The amount of change in the personnel.</td>
<td>7</td>
</tr>
<tr>
<td>Team Cohesion</td>
<td>The cooperativeness of the stakeholders.</td>
<td>9</td>
</tr>
<tr>
<td>Communication</td>
<td>The degree and efficiency of which information flows in the team.</td>
<td>4</td>
</tr>
<tr>
<td>Support for Innovation</td>
<td>To what degree assistance for new ideas is available.</td>
<td>1</td>
</tr>
<tr>
<td>Team Culture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capabilities and Experiences</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developer Temperaments</td>
<td>The mix of different temperaments on the team.</td>
<td>1</td>
</tr>
<tr>
<td>Analyst Capability</td>
<td>The skills of the system analyst.</td>
<td>8</td>
</tr>
<tr>
<td>Programmer Capability</td>
<td>The skills of the programmer</td>
<td>10</td>
</tr>
<tr>
<td>Applications Experience</td>
<td>The familiarity with the application domain.</td>
<td>7</td>
</tr>
<tr>
<td>Platform Experience</td>
<td>The familiarity with the hard- and software platform.</td>
<td>7</td>
</tr>
<tr>
<td>Language and Tool Experience</td>
<td>The familiarity with the programming language and tools.</td>
<td>8</td>
</tr>
<tr>
<td>Manager Capability</td>
<td>The control of the manager over the project.</td>
<td>7</td>
</tr>
<tr>
<td>Manager Application Experience</td>
<td>The familiarity of the manager with the application.</td>
<td>2</td>
</tr>
<tr>
<td>Environment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proper Workplace</td>
<td>The suitability of the workplace to do creative work.</td>
<td>3</td>
</tr>
<tr>
<td>E-Factor</td>
<td>This environmental factor describes the ratio of uninterrupted hours and body-present hours.</td>
<td>2</td>
</tr>
<tr>
<td>Time Fragmentation</td>
<td>The amount of necessary “context switches” of an employee.</td>
<td>1</td>
</tr>
<tr>
<td>Physical Separation</td>
<td>The team members are distributed over the building or multiple sites.</td>
<td>4</td>
</tr>
<tr>
<td>Telecommunication Facilities</td>
<td>Support for work at home, virtual teams, video conferencing with clients.</td>
<td>2</td>
</tr>
<tr>
<td>Project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schedule</td>
<td>The appropriateness of the schedule for the development task.</td>
<td>5</td>
</tr>
<tr>
<td>Requirements Stability</td>
<td>The number of requirements changes.</td>
<td>6</td>
</tr>
<tr>
<td>Average Team Size</td>
<td>Number of people in the team.</td>
<td>10</td>
</tr>
</tbody>
</table>

5. Related work

An early review of the state of the art in software development productivity was done by Jeffery and Lawrence [16]. They concentrated on the conflicting results w.r.t. some factors such as experience or size that in some studies were found to have a positive in others a negative effect. We do not consider that in our paper but only analyse the relevance of a factor in general.

Maxwell, Van Wassenhove and Dutta [21] relate their research on productivity factors in military software projects to earlier studies in other areas and the factors found there. However, this work is already 12 years old and a large number of studies have contributed to the knowledge about productivity factors since then.

Ramírez and Nembhard [25] analysed the more general category of knowledge worker productivity. Software developers are in their work part of these knowledge workers (KW) as opposed to manual workers. They state that “it seems to be of common agreement that to date there are no effective and practical methods to measure KW productivity.” Hence, they concentrate on a review of the dimensions used in the literature whereas our review considers the factors influencing productivity.

6. Conclusions

The productivity of the development team is decisive for successful software projects. Hatton [14] shows that there are large differences, especially in the abilities of the developers. “[...] in most experiments, analysts regularly record variations of a factor of 10 or more in the individuals’ performance.” This illustrates the large potential for improvements in development projects.

However, controlling productivity is only possible if the influencing factors are known. “You cannot control what you cannot measure.” [11] Hence, a clear list of influences on productivity in software development is needed in order to organise corresponding analysis and control activities. Existing productivity models and methods already make use of lists of productivity factors.

Yet, there is a large body of literature on productivity and productivity factors accumulated over the last decades. This paper provides a systematic review of this literature.
and a derived list of important factors based on their use in the studies. Soft and technical factors are investigated in equal detail and a list of factors is provided for each.

This list can now be used for modelling productivity and for productivity improvement methods. For example, the ProdFLOW\textsuperscript{TM} method described in [27] uses interview techniques for determining the most influential factors in productivity for a specific organisation. These interviews can be supported by the comprehensive knowledge about existing factors from the compiled lists.

For further research, we need to add further detail to the lists of factors by determining whether the factors influence productivity positively or negatively which is important for productivity models. Furthermore, there are influences between factors that can also have significant effects that need to be considered in this list and corresponding models.

Finally, for further surveys like this, it would be extremely useful if the researchers that report about the influence of specific factors on productivity were describing the factors, the measurement units and the context in more detail. Then the knowledge can be aggregated in ways that can provide even more value.

References

Comparative Study on Prediction Intervals of Effort Estimation Models Based on Linear Regression Models

Sousuke Amasaki
Comparative Study on Prediction Intervals of Effort Estimation Models Based on Linear Regression Models

Sousuke Amasaki
Department of Systems Engineering
Okayama Prefectural University
111 Kuboki, Soja, Okayama 719–1197, Japan
amasaki@cse.oka-pu.ac.jp

Abstract
OBJECTIVE – to evaluate different effort estimation models in terms of efficiency of prediction interval. It seems an important characteristic other than accuracy of single point estimate. However the past research has seldom evaluated a model in terms of that.

METHOD - we used the same experimental design, dataset, and performance measures in the past study. This study evaluated three linear regression models: log-normal, Poisson, and Gamma.

RESULTS – the Gamma regression model showed the highest efficiency of prediction interval. In terms of accuracy of single point estimate, the Gamma regression model and the Poisson regression model showed similar performance and were superior to the log-normal regression model.

CONCLUSIONS – first, an appropriate probability distribution improves efficiency of prediction interval. Second, high accuracy of single point estimate does not support high efficiency of the prediction interval. We also observed that the Gamma regression model was as efficient as an expert judgment-based method.

1. Introduction

Success of a software development project heavily relies on how exact an estimated effort is. Overestimates cause bidding with incompetent condition or failure of commitment. Even if stakeholders would commit to such a plan, its excessive budget must be wasted due to Parkinson’s Law [13]. On the other hand, underestimates impede project success. A project would not be able to keep up with an unrealistic plan based on an underestimate. This delay causes additional work to recover. It is also known that underestimates degrade product quality [4].

Effort estimation is a common research topic in software engineering. Especially, effort estimation models have widely been studied [5]. Many of them are based on statistics, data mining methods, and machine learning methods. With regard to objectivity, effort estimation models are preferable to expert judgment because they are far from external pressure and variability of individual skills and experience. Effort estimation models usually intake quantified characteristics of a project and estimate an effort for this project. Effort estimation models have been usually evaluated in terms of accuracy of single point estimate using empirical dataset. Common performance measures MMRE and PRED(25) are both calculated from absolute relative error of single point estimates.

On the other hand, practitioners estimate an effort in the form of an interval. A project involves high degree of uncertainty in early phase and the degree of uncertainty becomes low as this project proceeds. Thus, an interval is useful to represent an effort estimate exactly. The width of interval represents the degree of uncertainty. For example, a practitioner estimates a pair of minimum and maximum estimates of effort that is expected to include true effort in chance of 90%. This interval is called prediction interval. Prediction interval has more information than single point estimate. It is useful to evaluate an effort in relation to a probability of project success and to determine how much risks should be accepted.

While effort estimation based on the prediction interval is useful and popular in expert judgment-based methods, the past research has seldom evaluated effort estimation models in terms of efficiency of prediction interval. Jørgensen et al. defined performance measures for the prediction interval and used them to compare expert judgment-based methods with an effort estimation model based on a linear regression model [6]. However, we found no other study evaluating an effort estimation model using some such performance measure.

In this study, we evaluated and compared different ef-
fort estimation models in terms of efficiency of prediction interval. These effort estimation models are all based on linear regression models. They assume different probability distributions on effort. This evaluation was based on performance measures used in [6]. As a result, we quantitatively ascertained two things. First, an appropriate probability distribution improves efficiency of prediction interval even when a simple univariate regression model is used. Second, high accuracy of single point estimate does not support high efficiency on the prediction interval. We also observed that one model is as efficient as an expert judgment-based method.

The rest of this paper is as follows: Section 2 explains details of linear regression models we compared. Section 3 describes an experiment design, a dataset, and performance measures. In Section 4, the experiment results are examined. Section 5 shows an overview of the past research. Finally, Section 6 concludes this paper.

2. Linear Regression Models

2.1 Overview

This study evaluated and compared effort estimation models based on linear regression models for two reasons. First, they have widely been applied to effort estimation. Especially, their performance has often been used as baselines when a new effort estimation model is proposed and evaluated. We think it is useful that efficiency of their prediction interval can also be used as baselines for evaluation of a new effort estimation model. Second, linear regression models have an ability to represent uncertainty of an estimate. They are based on statistics which originally propose a notion of prediction interval. In case of linear regression models, a prediction interval is constructed from confidence intervals for coefficients and variance of a probability distribution of a dependent variable. This means that the difference of probability distributions influences a shape and a range of prediction interval.

The degree of reliability of a prediction interval is adjusted by confidence level. Confidence level is denoted by \((1 - \alpha)\) and typically expressed as a percentage \((1 - \alpha) \times 100\%\). Confidence level is usually set to typical percentages such as 90\%, 95\%, and 99\% similar to significance level in a hypothesis testing. Jørgensen et al. evaluated an ordinary regression model with 90\% prediction interval [6].

2.2 Regression Form

To now, several regression forms have been proposed to represent a relationship between effort and project characteristics [7]. Among these equations, one equation that relates log-transformed effort to log-transformed size-related project characteristic such as LOC and function points is widely used [2, 10]. This equation is as follows:

\[
\log(Effort) = \beta_0 + \beta_1 \log(Size). \tag{1}
\]

When canceling log-transformation, we get

\[
Effort = \exp(\beta_0)Size^{\beta_1}. \tag{2}
\]

Equation (1) is suitable for two reasons. First, non-linear relationship is assumed between effort and size-related project characteristics. This assumption is accepted though discussion still lasts. One example is the basic COCOMO which equation takes essentially the same regression form as equation (1). Second, equation (1) limits a value of effort to zero minimum. It is a desirable property of effort.

Since log-transformation alleviates heteroscedacity of effort distribution, it is easy to estimate coefficients by using an ordinary regression model which assumes normal distribution with constant variance. However, using this regression model with equation (1) has a problem that a probability distribution on effort is limited to log-normal distribution. This limitation is unsuitable for our study because this study tries to evaluate linear regression models based on different probability distributions on effort.

2.3 Probability Distribution on Effort

This study employed generalized regression models (GLMs). GLMs allow some kinds of non-linear relationships between a dependent variable and independent variables, and some probability distributions other than normal distribution. GLMs, in contrast to ordinary regression model, estimate coefficients directly from equation (2). We compared three probability distributions: log-normal, Poisson, and Gamma distributions. These probability distributions are assumed on effort. They have different characteristics as follows:

**Log-normal**  Log-normal regression models have widely been used in studies on effort estimation models. When log-normal distribution is log-transformed, it becomes identical to normal distribution. The mean and variance of a random variable \(X\) with log-normal distribution are, respectively,

\[
E[X] = \exp(\mu + \frac{\sigma^2}{2}),
\]

\[
Var[X] = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)
\]

where, \(\log(X) \sim N(\mu, \sigma)\).

Log-normal distribution is skewed right and long-tailed. This property meets a characteristic of software development effort that at small probability, completing a task consumes great amount of effort.

2
Poisson  Poisson distribution is usually applied to modeling counts of events occurred within a certain duration or area. Poisson distribution always takes positive and its variance is proportional to a mean. These properties are suitable to model heteroscedacity of effort. Thus, it was sometimes employed to effort estimation model[2]. The mean and variance of a random variable $X$ with Poisson distribution are, respectively,

\[
E[X] = \lambda,
\]
\[
V[X] = \lambda
\]

where, $\lambda$ is a parameter of Poisson distribution.

Gamma  Gamma distribution is used to model waiting time or service time in queuing theory. Kitchenham et al. pointed out that this characteristic is appropriate for modeling a time of completing a task, and they argued how to apply Gamma distribution to effort estimation [8]. Gamma distribution always takes positive and its standard deviation is proportional to a mean. A shape of Gamma distribution is skewed right and long-tailed as same as log-normal distribution. Gamma distribution takes shape parameter $\alpha$ and scale parameter $\lambda$. The mean and variance of a random variable $X$ with Gamma distribution are, respectively,

\[
E[X] = \frac{\alpha}{\lambda},
\]
\[
V[X] = \frac{\alpha}{\lambda^2}
\]

3. Experiment

3.1 Performance Measures

This study evaluated three linear regression models described in Section 2 in terms of efficiency of prediction interval. We also evaluated them in terms of accuracy of single point estimate because we think both characteristics are important. For prediction interval, two performance measures were employed: HitRate and medianPIWidth. Both performance measures were also employed in [6].

HitRate measures how exactly prediction interval covers the corresponding actual value. The definition of HitRate is as follows:

\[
HitRate = \frac{1}{n} \sum_{i=1}^{n} h_i,
\]

\[
h_i = \begin{cases} 
1 & \text{min}_i \leq \text{Act}_i \leq \text{max}_i, \\
0 & \text{Act}_i > \text{max}_i \cup \text{Act}_i < \text{min}_i
\end{cases}
\]

where, min$_i$ and max$_i$ represent lower-limit and upper-limit of $(1 - \alpha) \times 100\%$ prediction interval estimated for i-th test-case. Act$_i$ represents the actual effort of i-th test-case. HitRate is expected to match with the confidence level $(1 - \alpha) \times 100\%$. In case of 90% prediction interval, it is too narrow if HitRate shows 0.6(60%), and too wide if 0.99(99%).

medianPIWidth measures how efficient prediction interval is. It is a median of PIWidths of all test-cases. PIWidth is a relative width of a prediction interval. The definition of PIWidth$_i$ for i-th test-case is as follows:

\[
PIWidth_i = \frac{\text{max}_i - \text{min}_i}{\text{Est}_i}
\]

where, Est$_i$ denotes the estimated effort for i-th test-case. An effort estimation model with efficient prediction interval shows small medianPIWidth. When two models show similar HitRate, the one model showing smaller medianPIWidth is better than the other.

For single point estimate, five performance measures were employed: medianMRE, PRED(25), medianBRE, BPRED(25), and sd(BRE). The first two measures are based on MRE and the other measures are based on BRE. MRE and BRE are performance measures for a test-case. They are defined for i-th test-case as follows:

\[
\text{MRE}_i = \frac{\|\text{Act}_i - \text{Est}_i\|}{\text{Act}_i}
\]
\[
\text{BRE}_i = \frac{\text{Act}_i - \text{Est}_i}{\text{min}(\text{Act}_i, \text{Est}_i)}
\]

Here, min$(a, b)$ is a function returning smaller one between $a$ and $b$. medianMRE is a median of MREs. This is widely used in studies of effort estimation models. meanMRE or MMRE is usually used for evaluation but has not been used in this study because the number of test-cases is small and meanMRE is sensitive to an outlier in this situation. PRED(25) measures how closely single point estimate is concentrated to the corresponding actual value. PRED(25) is defined as a ratio that single point estimates for test-cases satisfy $\text{MRE}_i \leq 0.25$.

medianBRE and BPRED(25) correspond to medianMRE and PRED(25), respectively. medianBRE is a median of BREs. BRED(25) is defined as a ratio that single point estimates for test-cases satisfy $\|\text{BRE}_i\| \leq 0.25$, sd(BRE) measures a standard deviation of BREs. It is a performance measure similar to BRED(25).

BRE revises asymmetry between MRE of underestimate and overestimates. MRE takes at most 1 when an effort is underestimated but takes easily more than 1 when the effort is overestimated. On the other hand, BRE bases at 0 and takes positive when an effort is underestimated, and negative when the effort is overestimated. For instance,
when a single point estimate results in a one-third of the actual effort, \( BRE \) and \( MRE \) are 2 and \( \frac{2}{3} \), respectively. When a single point estimate results in three times of the actual effort, \( BRE \) and \( MRE \) are \( -2 \) and 2, respectively. Thus, \( medianBRE \) measures whether an effort estimation model tends to overestimate or to underestimate.

### 3.2 Procedure

We adopted the same procedure described in [6] because following this procedure makes it possible to compare linear regression models to an expert judgment-based method in [6]. The procedure consists of three steps as follows:

**Step 1** Learn each linear regression model with Task 1 to Task \((i - 1)\).

**Step 2** Estimate single point estimates and 90% prediction intervals for Task \(i\) by applying the learned linear regression models.

**Step 3** Repeat Step 1 to Step 2 until \(i\) reaches the number of dataset.

In this experiment, \(i\) starts from 11 because it was assumed that Task 1–10 have been completed.

While Jørgensen et al. calculated a prediction interval based on a formula defined in statistics, we employed a simulation-based method which a package \texttt{arm} implements for statistics software R.

### 3.3 Dataset

We used a dataset shown in [6]. This dataset consists of 15 tasks in maintenance phase of multiple software applications. While some applications provide multiple maintenance tasks, we did not consider the difference of applications. As same as an experiment in [6], these tasks are arranged and Task 11–15 are used as test-cases. This dataset includes effort, size of new code, modified code, and deleted code. In [6], sizes of all kinds of code are summed and log-transformed. This new metric was then used as an independent variable of a univariate linear regression model. We also used this formulation.

Figure 1 shows a scatterplot of effort and summed code size. Symbols assigned for Task 11–15 are unique while those for Task 1–10 are identical. Most of the tasks imply somewhat relationship between effort and summed code size but some tasks look like outlier.

### 4. Results

Table 1 and 2 show single point estimates and 90% prediction intervals for Task 11–15, respectively. Table 1 shows that linear regression models estimated different widths of prediction intervals. The log-normal regression model estimated the widest prediction interval and the Poisson regression estimated the narrowest prediction interval. Table 2 indicates that single point estimates by the log-normal regression model was always smaller than those by the others. The Gamma regression model and the Poisson regression model did not show apparent difference in single point estimates. From Table 1 and 2 we confirmed that linear regression models estimated different single estimates and prediction intervals.

Table 3 shows performance measures of linear regression models. In addition, fourth row “EMP+Human” indicates performance measures for an expert judgment-based method in [6]. We first determined that the Gamma regression model showed the highest efficiency of prediction interval. From Table 3, we found that the log-normal regression model and the Gamma regression model showed sufficient \( HitRate \) (80%) against 90% confidence level. On the other hand, the Poisson regression model showed insufficient \( HitRate \) (40%). Table 3 shows that \( medianPIWidth \) of the Gamma regression model was smaller than that of the log-normal regression model. Since these regression models showed the same \( HitRate \), it can be said that the Gamma regression model estimated more efficient prediction interval than the log-normal regression model. Also, \( Hitrate \) and \( medianPIWidth \) of the Gamma regression model are nearly the same as those of an
expert judgment-based method. That is, the Gamma regression model was as efficient as the expert judgment-based method.

Next, we determined that the Gamma regression model and the Poisson regression model showed similar performance and were superior to the log-normal regression model in terms of accuracy of single point estimate. Table 3 indicates that the log-normal regression model showed the best medianMRE though all linear regression models showed identical PRED(25). On the other hand, the log-normal regression model showed the worst medianBRE and BPRED(25). The other BRE-based performance measures also indicated the same results. In terms of accuracy of single point estimate, the Gamma regression model and the Poisson regression model always showed nearly identical performance. These inconsistent results regarding the log-normal regression model are due to the difference between definitions of MRE and BRE. While MRE penalizes an overestimate more seriously than an underestimate, BRE penalizes them equally on the basis of multiplicative form. Table 2 shows that the log-normal regression model tends to underestimate in comparison with the others. Thus, the log-normal regression model showed better medianMRE, and worse medianBRE and BPRED(25) than the others. Taking into account that underestimates are considered more seriously than overestimates [9], we accepted that BRE-based performance measures are more suitable than MRE-based performance measures. Finally, we concluded that the Gamma regression model was the most appropriate model.

As a result, we quantitatively ascertained two things. First, an appropriate probability distribution improves efficiency of a prediction interval even when a simple univariate regression model is used. In this study, the Gamma regression model showed the best performance among examined models. Second, high accuracy of single point estimate does not support high efficiency of prediction interval. In this study, the Gamma regression model and the Poisson regression model showed nearly identical results in terms of accuracy of single estimate. However, they show different results in terms of efficiency of prediction interval.

5. Related Work

Effort estimation models have widely been studied. A review paper on the area of effort estimation [5] classified research papers published before 2004 into nine research types. Following this classification, sixty one percents of these papers proposed or evaluated new effort estimation models. This review also classified effort estimation approach into twelve types. Following this classification, more than 70% of research papers proposed effort estimation models based on statistics, data mining, and machine learning methods. While it was said that an estimate is a probabilistic assessment of a future condition [8], effort estimation uncertainty has not been popular research issue [12]. In the past research, effort estimation models have usually evaluated in terms of performance of single point estimate.

Some effort estimation models are based on a method that can estimate uncertainty of prediction. Statistical approach such as linear regression models can originally estimate prediction interval as shown in this study. Bayesian statistics has been applied to machine learning methods such as neural network and they now can estimate predictive interval [3]. In the same way, effort estimation models based on Bayesian statistics such as Bayesian network [11] will also estimate uncertainty. When an effort estimation
Table 3. Performance Measures for Task 11–15

<table>
<thead>
<tr>
<th>Method</th>
<th>HitRate (90%)</th>
<th>medianPIWidth</th>
<th>medianMRE</th>
<th>PRED(25)</th>
<th>medianBRE</th>
<th>sd(BRE)</th>
<th>BPRED(25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-normal</td>
<td>80</td>
<td>5.2</td>
<td>0.30</td>
<td>0.4</td>
<td>0.16</td>
<td>2.3</td>
<td>0.2</td>
</tr>
<tr>
<td>Poisson</td>
<td>40</td>
<td>0.6</td>
<td>0.78</td>
<td>0.4</td>
<td>-0.01</td>
<td>2.1</td>
<td>0.4</td>
</tr>
<tr>
<td>Gamma</td>
<td>80</td>
<td>2.9</td>
<td>0.78</td>
<td>0.4</td>
<td>-0.05</td>
<td>2.0</td>
<td>0.4</td>
</tr>
<tr>
<td>EMP+Human</td>
<td>82</td>
<td>2.8</td>
<td>0.50</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

model is based on no statistical basis like analogy-based method, this method needs to use a technique such as bootstrap [1].

While major methods employed in effort estimation models can estimate uncertainty, we could not find comparative study on effort estimation models in terms of estimation uncertainty. Thus, we think that this study contributes to research on effort estimation uncertainty.

6. Conclusion

This study evaluated effort estimation models based on linear regression models in terms of efficiency of prediction interval. As a result, we quantitatively ascertained two things. First, an appropriate probability distribution improves efficiency of a prediction interval. Second, high accuracy on a single point estimate does not support high efficiency on a prediction interval. We also observed that the Gamma regression model is as efficient as a kind of expert judgment.

The result of this study heavily depends on dataset. In theoretical terms, there is no superiority of Gamma distribution to log-normal distribution while Poisson distribution is inferior to them. They are often interchangeable and thus we may find no difference when using other dataset. Thus, experiments with many larger datasets should be needed to examine whether effort distribution is preferable to Gamma distribution in general situations.

This study dedicated on linear regression models but we also need to compare effort estimation models based on other methods such as neural network and analogy-based method, as same as ordinary comparative study.

References

Using a Bayesian Network in the ProdFLOW Approach

Stefan Wagner, Melanie Ruhe
Using a Bayesian Network in the ProdFLOW™ Approach

Stefan Wagner
Technische Universität München
Garching b. München, Germany
wagnerst@in.tum.de

Melanie Ruhe
Siemens AG
Munich, Germany
melanie.ruhe@siemens.com

Abstract

ProdFLOW™ is a new approach for the productivity analysis and management of research & development organisations created by the research department of the Siemens AG. Its core are organisation-specific models based on the respective substantial levers of productivity. Levers that are both influenceable and measurable are compiled together with the experts of the organisation. This paper proposes to use Bayesian networks for building such models. It is shown how the networks are structured, how they are parameterised and used to analyse different improvement scenarios. Experiences from a case study suggest that Bayesian networks are a suitable technique for the organisation-specific models in ProdFLOW™.

1. Introduction

Research and Development (R&D) activities are strongly affected by the human factor and dominated by cognitive activities and knowledge work, no matter whether its software, hardware or system development. This is a major difference to the manufacturing business. Usually the input and output of the R&D process differ from one R&D project to another and the R&D process is unique, i.e. not directly and completely repeatable. That means existing approaches of measuring and improving productivity in manufacturing need to be adapted to the characteristics of R&D. In the research department of the Siemens AG a new approach for the management of productivity in the area of R&D is being created. Former studies often start with fixed models for the productivity and try to calculate quantitatively the relation between productivity and its influencing factors by analysing regression models. This procedure presupposes that productivity is significantly determined by these influencing factors and that these indicators can easily be changed by the organisation. However, we have made the experience that the so-called typical productivity factors are not that typical within Siemens. Therefore, we depart from the fixed model approach, which might not fit to the conditions of the organisation, and develop a new approach called ProdFLOW™.

Problem To build a new and specific model for each organisation is potentially a very elaborate and difficult task. Also the high uncertainty in the factors and their interrelationships render their results potentially unreliable. The problem we address in this paper is how to build organisation-specific productivity models in a way that the effort is justifiable and their results are still useful.

Contribution We describe how Bayesian networks can be used as a modelling technique for organisation-specific productivity models. This technique fits well because of its suitability for modelling influences between levers as well as modelling uncertainties. The latter are mainly introduced by the questionnaires that are used to fill the model with values. We show the validity of the approach by a case study at Siemens.

Organisation We start by discussing generally what productivity in R&D businesses means in Sec. 2 and summarise the Siemens ProdFLOW™ approach in Sec. 3. The Bayesian network approach for productivity models is proposed in Sec. 4 and the corresponding case study is described in Sec. 5. Conclusions are given in Sec. 6. Related work is cited where appropriate.

2. Productivity in R&D businesses

The well-known definition of productivity (relation of output quantity to the input quantity) is often disputed in the world of R&D. Due to the diversity of disciplines that use the term productivity, there is no clear cut definition of productivity and related terms. This lack of common agreement on what constitutes productivity is perceived as a major obstacle for a substantiated discussion of productivity. Product management aims to release as early as possible, in order to maximise the relative market value, whereas the product development team wants to maximise the creation
of value in the sense of the fulfilment of all customer requirements [1, 3]. The motto deliver value in time motivates effective (value) and efficient (in time) product development. Thus, we define productivity in R&D as the relation between value creation for the customer (output) and the effective budget for research and development (input):

\[
\text{Productivity} = \frac{\text{R&D value creation}}{\text{Effective R&D budget}} \quad (1)
\]

Productivity increases if more or better products are developed from the same resources. Better products may be products of higher quality, higher reliability or flexible products (=higher value for the customer) and thus the created value (for the customer) increases. If we develop the same products with fewer resources, productivity may also increase.

3. ProdFLOW™

ProdFLOW™ stands for “Productivity in R&D with FLOW”. Especially in the context of knowledge work the status of flow [3] should increase productivity. Within ProdFLOW™ we focus on specific levers when improving productivity in terms of increased value creation and emphasise this using the abbreviation FLOW, which stands for “Focus on Levers to optimise your Work”. More details on the approach in general can be found in [9].

The procedure can be applied both to small and large R&D organisations and is also scalable to single phases of the development process. Important is, that the results are developed individually for the evaluated organisation and that it is thus not transferable to other organisations. The aspect of comparability is excluded from this approach because we consider it more important to improve individual productivity than to compare it. We split up the approach into four working steps and the preparation step 0 “Customise Analysis”. Those steps, if necessary, can be iterated several times. The main objective is to improve the identified, major/top levers and to lift these into a balanced condition. In the following these individual steps of our approach are described with the respective activities, tasks and results.

3.1. Step 0 – Customise analysis

Step 0 of ProdFLOW™ has the objective to prepare and plan the subsequent steps. Therefore, the basic goals and characteristics, economic data and future strategic plans of the organisation are analysed. A stakeholder diagram is elaborated to get indications for potential areas of major productivity levers by understanding the network of internal and external stakeholders and their expectations, impacts, cooperation as well as priorities.

3.2. Step 1 – Identify productivity levers

Step 1 has the objective to identify and define the substantial levers with influence on productivity. For the collection of data we employ individual interviews. We make use of an interview guideline for the preparation of the interviews and as a starting point into the productivity analysis. The interview itself is an open but guided conversation with the focus on how value is created for the customer of the analysed organisation. Important is also to query facts and no opinions or rumours. In the follow-up to the interview, minutes with the major information of the interview are developed, i.e. the logging represents an interpretation. Afterwards the logging minutes are provided to the interview partner for authorisation. That gives the evaluation the guarantee that every analysis is based on the right facts. This procedure is based on the work of [1].

Finally a list with levers and their definition based on the aggregated results of the analysis is created. This final activity is critical based on our experience, since the definition of the levers must be formulated objectively and be generally understandable.

3.3. Step 2 – Rank and filter levers

Step 2 has the objective to rank the identified levers regarding the criteria importance and improvement potential as well as to filter the levers according to the criteria measurability and influenceability. The results are collected, cumulated and evaluated in the sense of an average ranking, i.e. average rank and the standard deviation for each lever are calculated. In the case of high standard deviation, i.e. very different opinions, the results must be discussed in the organisation to be clarified.

The results are visualised in a so-called prioritisation matrix to present the ranking. The two dimensions of the matrix represent (a) the importance of the lever and (b) its improvement potential. Levers, which are regarded as important as well as having high potential for improvement, are located in the matrix in the upper right quadrant.

In a further step the levers with high priority are filtered regarding to the criteria measurability and influenceability. Levers, which cannot be influenced by the organisation or cannot be measured, will be marked and not further considered in the next steps of the approach.

3.4. Step 3 – Define lever indicators and initiate improvement

Step 3 has the objective, to evaluate and measure the identified top levers and to initiate a coordinated productivity improvement project. This is focused on the top levers
that considerably affect productivity. To determine suitable measurement or evaluation instruments measurement expertise is required. The identified levers usually are not standard factors that can be looked up in some reference textbooks. In parallel to the definition of the measurement and evaluation instruments, the organisation has to define measures to improve the levers. This can include a first estimation of the current status of the lever based on the defined measurement instruments.

3.5. Step 4 – Measure levers and determine balance

Step 4 has the objective to track the progress of the productivity improvement project as well as to analyse the balance of levers. The balance is the precondition for the increased value creation at the customer and the related increased productivity. The idea of balance in the context of ProdFLOW™ is a new concept which tries to analyse and understand the dependencies and influences of the productivity levers on each other as well as on productivity itself. The idea is in line with the concepts of Pareto optimality or Pareto efficiency: An economic system that is Pareto efficient implies that no individual (lever) can be made better off without another being made worse off [6]. Within our approach it means, that there is no reason to improve one single lever if another important lever may deteriorate. Due to the fact, that the levers may have a mutual influence the direction (positive, negative) as well as the intensity (high, medium and low) of the influence has to be analysed.

3.6. Model Requirements

Step 4 of ProdFLOW™ is the one we concentrate on in the following. So far no explicit models of productivity have been created. The models in this step need to allow an analysis, simulation, and balancing of the productivity levers. From these goals, we can derive a set of requirements on the modelling technique that we use for modelling and analysing the productivity factors.

First, we believe that there are significant influences in between levers that are partly contrary. Hence, it is necessary that the modelling technique allows to model those influences. Second, the productivity factors are diverse and can be measured in various kinds of ways in different scales and units. Therefore, the modelling technique needs to be able to handle those different kinds of data and their combination. Third, for several of the identified productivity factors, it will be necessary to determine their value by expert opinion. This and other measurement methods introduce uncertainty in the data. The modelling technique is required to be able to represent that uncertainty. Fourth, a graphical notation is seen as helpful in order to handle the complex interrelationships. Finally, the effort for developing the model itself needs to be small so that the benefits outweigh the costs.

4. Bayesian network model

We first introduce briefly what Bayesian networks are and how they can be used to model productivity. We explicitly address the difficult issue of probability elicitation for the model and provide the assumptions made in the model.

4.1. Bayesian networks

Bayesian networks, also known as Bayesian belief nets or belief networks, are a modelling technique for causal relationship based on Bayesian inference. It is represented as a directed acyclic graph (DAC) with nodes for uncertain variables and edges for directed relationships between the variables. This graph models all the relationship abstractly. For each node or variable there is a corresponding node probability table (NPT). These tables define the relationships and the uncertainty of these variables. The variables are usually discrete with a fixed number of states. For each state, the probability that the variable is in this state, is given. If there are parent nodes, i.e. a node that influences the current node, these probabilities are defined in dependence on the states of these parents. An example is shown in Tab. 1. There the variable is for example with a probability of 70% in the state low if both parents are in the state low, and with 55% in low if the first parent is in high and the second is in low.

<table>
<thead>
<tr>
<th>Variable</th>
<th>low</th>
<th>med</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>low</td>
<td>low</td>
<td>0.7</td>
<td>0.65</td>
</tr>
<tr>
<td>high</td>
<td>0.3</td>
<td>0.35</td>
<td>0.6</td>
</tr>
</tbody>
</table>

With respect to the above defined requirements on the modelling technique for productivity analysis, we can conclude that Bayesian networks are able to model influences between levers. However, these influences can only exist in one direction. Mutual influences are not possible. Different scales and units can be used as long as there is a way to discretise them. Finally, uncertainty is a first-level citizen in Bayesian networks. Hence, we believe that they are a suitable basis for the models used in our methodology.

Moreover, Stamelos et al. [10] have already used Bayesian belief networks for productivity prediction. They employed the COCOMO I factors and combined them in
such a network. They conclude that this is a suitable approach especially when expert judgement has to be included.

4.2. Building a Network

We assume that we have from the first two steps the 3–5 identified top levers from an organisation as input. We only consider those, and not all identified levers, in the network. The reason is the effort for thoroughly modelling these factors and elicit the probabilities. As is stated in [7] “In order to reduce the size of the elicitation task, one should seek to consider only those variables and their values that are absolutely necessary for the specific problem.” For these levers, the influences between them need to be included.

From each of these levers, there is an edge to Productivity which is measured in low, medium and high. Furthermore, it is common to employ node divorcing in order to have not more than two parents per node. If there are more parents, one or more additional nodes can be included in between to reduce the number of parents. Especially for the relation to Productivity, this can be employed. However, in our case, usually it is not necessary because we use a simple combination of the input nodes. We also introduce the additional node Other influences that accounts for all the other productivity levers not explicitly modelled. This “trick” avoids that a single or very few levers can have a too huge influence on the overall productivity. The complete standard pattern for the used models is shown in Fig. 1.

![Figure 1. General model structure](image)

4.3. Probability Elicitation

It is an acknowledged problem in Bayesian networks that the elicitation of the probabilities for the node probability tables (NPT) is difficult and elaborate [4]. This is especially the case when the figures need to be estimated by an expert. First, the number of estimates increases exponentially with the states of the predecessor nodes. Second, people are bad at estimating probabilities in general [2]. As in our method probably most of the identified levers need to be estimated with expert opinion, we need clear guidelines for the probability elicitation.

There are several general guidelines w.r.t. eliciting expert judgement [2]. The elicitation questions must be clearly expressed and validated with test runs in order to minimise misunderstandings. Furthermore, it turned out that the expert should know how the responses will be processed. This helps them to give a more accurate estimation.

Various methods have been proposed to mitigate these problems. However, as is discussed for example in [11], they all have drawbacks. A graphical probability scale allows the expert to mark they estimated probability on a horizontal or vertical line, usually marked with 0, 50 and 100. The support from the method is conceived as rather low [11] and the so-called spacing effect occurs, i.e., the experts tend to organise the probabilities in a way to increase visual attractiveness. It is also often acknowledged [2] that it is easier to think in frequencies instead of probabilities. This means it is cognitively easier to imagine that something holds for 5 out of 100 people than for 5%. Yet, in [11] it is reported that the experts had difficulties using frequencies for rare combinations. Finally, gambles are also proposed for a more accurate judgement. However, gambles are demanding for the experts and take significant time. What seems to be practically useable is a probability scale with both numbers and words [8].

Fenton et al. [5] employ a truncated normal (TNorm) distribution as basis to generate the corresponding probabilities in an NPT. The experts describe their intuition first for extreme points in a kind of “truth table” and then calibrate the corresponding distribution. This is the state of the art in defining probabilities for Bayesian networks. We will use this technique as implemented in the AgenaRisk tool. However, this still leaves us with the task of defining the mean and variation of the TNorm distribution $\text{TNorm}(\mu, \sigma^2)$. For this, we will build on the influences. For all values to be determined, we use expert opinion asked in the form of a questionnaire. The complete elicitation is structured in five parts.

4.3.1 Part I – Lever influences

First we need to determine which levers are influenced by others. This is done by asking for each pair of levers whether they influence each other. If an influence given in both directions, the stronger influence is included in the model. In Fig. 1 $L2$ influences $L1$ and $L3$, and $L3$ also influences $L1$. 

- 31 -
4.3.2 Part II – Independent levers

Next, we look at all lever nodes that do not depend on another lever. In the network from Fig. 1, this is the case for \( L2 \). As in most cases, these nodes will not have a natural distribution, we assume a uniform distribution. This is the standard procedure if no further knowledge is available.

4.3.3 Part III – Dependent levers

The third part contains the determination of the NPTs for the levers that are influenced by other levers. First it has to be determined if the influence is positive or negative, then the strength of the influence needs to be set. We do this in the questionnaire already when asking which levers do influence each other by asking for each pair if there is a light, medium or strong influence. From this the mean \( \mu \) of the TNorm distribution is derived. This could also be enriched with weights as described in part IV below. For the variance, we need an additional question: How confident are you about the influence from Lever 1 on Lever 2 on a scale from 0 to 10? This question could be aided by a scale with numbers and words describing what that valuation is supposed to mean, for example from neglectable to directly determined.

4.3.4 Part IV – Productivity

The NPT of the Productivity node is determined accordingly. We again use the mean of all levers (using the \( 1 - l \) normalisation if necessary) in order to determine productivity. The variation is similar to above determined. In this part, it is especially important to add weights to the mean calculation. It is obvious that not all levers will contribute with the same intensity to productivity. This needs to be reflected in weights. We ask the experts to assign 100 points on the influencing factors depending on the strength of the influence. These points can then be used as weights and the Bayesian network is fully completed.

4.3.5 Part V – Current state

For each lever, a set of questions is derived that determines its current state. For example, for a lever called storing and finding knowledge we used as one question “How satisfied are you with the current knowledge management system?” The answers of these questions are all on a scale from 1 to 6. In the Bayesian network (Fig. 1) each of the questions is modelled as a separate node as indicated by three example nodes. Their joint influence on each lever determines the current state of the lever in the analysed organisation.

4.4. Assumptions

For the above sketched approach, we use several assumptions to make it practical. We briefly discuss these assumptions in the following.

1. The first assumption is that there is always a dominate influence in one direction from one lever to the other. Bayesian networks do not allow cycles in the graph. Hence, it is not possible to model influences in both directions between two levers. However, from the experiences with the levers we found in the interviews, we are confident that it is sufficient to have only an influence in one direction.

2. The second assumption is that we use a uniform distribution for the node states of nodes that have no parent (i.e. no levers that influence them). A prior distribution does only make sense if there is a natural distribution available, possibly determined empirically. For most levers, this distribution will not be available because of the lack of information for that specific department. A standard statistics procedure is then to use the uniform distribution.

3. Using the TNorm distribution for the probabilities in a node with parents. Fenton, Neil and Galan Caballero [5] describe this approach in detail. The idea is that having ranked nodes (i.e. nodes in which the states have a ranking order), the experts find it easier to give the central tendency of the node based on the value of the influencing node. However, this relationship is not completely certain and hence needs an uncertainty distribution around it. This is similar to linear regression where a normal distribution is used to model the uncertainty. For the ranked nodes, this is only changed to the doubly truncated Normal distribution that is only defined in the \([0, 1]\) region. Hence, “This enables us to model a variety of shapes, including a uniform distribution, achieved when the variance \(\sigma^2 \to \infty\), and highly skewed distributions, achieved when \(\sigma^2 \to 0\)” [5]

5. Case Study

The first steps of the ProdFLOW™ approach have been used in several case studies now. We went back to one of case studies done with a Siemens department and added the next step of model building. A questionnaire was prepared with the questions to determine the NPTs and the current state as described above.

In order to then validate whether the model actually describes reality in an acceptable way, the model derived from
the questionnaire was presented to 5 volunteers that provided their expert opinion. It is analysed whether the output of the model corresponds to the expectation of the experts. For this, we used 7 scenarios in the model that were rated by the experts on 6-point scale from “Does barely meet expectation (1)” to “Does completely meet expectation (6)”. The results are shown in Tab. 2. It shows that for nearly all of the scenarios the evaluation lies between 5 and 6 which represents a high correspondence with the expectations. Only The scenario in which one lever was strongly improved, the result is poorer. For this scenario, we repeatedly got the comment that the effect of a single lever should not be that strong.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Med.</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Baseline”</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6.0</td>
</tr>
<tr>
<td>“Very good”</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>5.0</td>
</tr>
<tr>
<td>“Very bad”</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>6.0</td>
</tr>
<tr>
<td>“Improving L1”</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6.0</td>
</tr>
<tr>
<td>“Improving L2”</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>5.0</td>
</tr>
<tr>
<td>“Improving L3”</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5.0</td>
</tr>
<tr>
<td>“Strong L3”</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Another frequent comment was that the lever L2 should have a stronger effect than the lever L3. In the model it is the other way round as derived from the questionnaire.

The validation showed mainly three lessons:

- The model in general meets the expectations of the experts.
- There should be an additional factor Other influences that has an effect on productivity so that a single lever has less influence. This is already incorporated in the description in section 4.
- The way the questionnaire asked for the influence weights was not optimal as the experts agreed on a different ranking that the questionnaire average.

Possible alternatives to the simple questionnaire would either be a graphical representation that fosters intuitive understanding or a coached, workshop-like interview.

6. Conclusions

ProdFLOW™ is a new approach to productivity analysis for R&D organisations based on the assumption that there cannot be a fixed model of productivity factors valid for all. Therefore, the approach contains organisation-specific models that contain only the most relevant productivity levers. It is challenging to find a suitable modelling technique that provides the necessary mechanisms and most importantly allows an efficient and effective creation of such models.

We propose to use Bayesian networks in such organisation-specific networks because they are able to handle influences between levers, they can work with different scales and units, and they directly support to model the uncertainty in the data. We employed the truncated normal distribution approach [5] for an efficient determination of the needed data that was elicited by a questionnaire. We showed in a case study with a department of Siemens that it is practically possible to build such a model and that it fits well to the experts expectations.

References

Lessons Learned in Evaluating Software Engineering Prediction Systems

Wasif Afzal, Richard Torkar
Lessons from applying experimentation in software engineering prediction systems

Wasif Afzal, Richard Torkar
Blekinge Institute of Technology,
S-372 25 Ronneby, Sweden
{waf,rto}@bth.se

Abstract

Within software engineering prediction systems, experiments are undertaken primarily to investigate relationships and to measure/compare models’ accuracy. This paper discusses our experience and presents useful lessons/guidelines in experimenting with software engineering prediction systems. For this purpose, we use a typical software engineering experimentation process as a baseline. We found that the typical experimentation process in software engineering is supportive in developing prediction systems and have highlighted issues more central to the domain of software engineering prediction systems.

1 Introduction

Software development process is costly and therefore demands efficient allocation of resources. Measurement during the different phases of a software development process makes different activities visible and hence provides opportunities for making efficiency gains, not necessarily limited to resource allocation. There has been significant interest in the software engineering research community to make use of these measures for predicting the future outcomes, in the form of prediction systems and models. Models act as a substitute for the complex real-world systems, to help us better represent the realities [12]. Statistical models are more common to software engineering whereby they are utilized for prediction of probabilistic future behavior of a system using prior data and extrapolation or interpolation of data based on a mathematical fit. There have been many published results on software effort/cost estimation and software fault/fault count predictions. With so many studies, the problem of reliable predictions is still largely unsolvable as we are not able to reach a general conclusion due to contradicting empirical results. This shows that there is enough uncertainty in the software prediction process that hampers reaching consistent and reliable results. One of the obvious reasons for having such a variation in the prediction results is that the software development process is seldom repeatable. Each new software project is innovative and many times solves new problems. Therefore, we have to deal with complex relationships among several variables and the interaction of application with its environment [14]. However, this is not the only problem facing inaccurate predictions in software engineering.

This study, in addition to summarizing some of the key reasons for inaccurate and contradictory prediction results, presents lessons/guidelines based on our experience in experimenting with prediction systems and related literature investigation. We use the typical experimentation process (given in [35]) as a baseline to the process of building software engineering prediction systems and to present critical issues specific to software prediction studies. The motivation for using a process-centric approach is that the research procedure quality is bound to impact the conclusions of a study. Myrtveit et al. [26] argues that lack of convergence of studies on software prediction models is partly attributed to variation in their quality of execution. Therefore, by improving the experimentation process we can achieve better convergence of results and improve validity of conclusions.

2 Experimental software engineering

Experiments in software engineering is part of a wider context i.e. empiricism in software engineering. The important reasons for undertaking quantitative empirical studies (i.e. experiments and case studies) are summarized by Wohlin et al. [35] as, “to get objective and statistically significant results regarding the understanding, controlling, prediction and improvement of software development”.

In software engineering, the steps required to perform experiments have been documented in a dedicated book to help software engineers in performing experiments [35]. According to this book, the steps constituting the process
of experimentation include: definition; planning; operation; analysis and interpretation; presentation and package.

The representation of the experimental process in above steps manifest that experimentation is a formal and controlled activity [35]. There are few studies that report experimentation in software engineering, Sjøberg et al. [31] found 103 papers out of 5453, taken from 12 conferences and journals, that could be categorized as being experiments [18]. Similar results are also reported by [33].

Following is a summary of different experimental steps [35]:

Definition: The definition step helps defining the goals and objectives of the experiment. This is one of the foundation steps for experimentation. Planning: The planning step includes determination of experiment context, formal statement of hypothesis, selection of variables and subjects, selecting experimental design, instrumentation and validity evaluation. Operation: The experiment operation consists of preparation, execution and data validation. Analysis and interpretation: The first step in analysis is to use descriptive statistics to provide a visualization of data. The second step is data reduction and the third step is hypothesis testing. Presentation and package: This step concerns the documentation of the experimental process and final results.

We use this experimentation process as a baseline to present the association with prediction studies in software engineering.

3 Software engineering prediction systems

There are many opportunities of making use of prediction systems in software engineering. Fenton and Pfleeger [11] show that predictions are needed throughout the software development life cycle, from feasibility through maintenance. The two most commonly targeted areas of prediction in software engineering are project effort and faults [29].

Software engineering literature has established several benefits out of accurate predictions and estimates:

1. Accurate cost estimation eliminates chances of overrun budgets and schedules. Similarly, overestimation can be avoided to achieve time and resource efficiencies [16].

2. Timely identification of fault-prone modules assists in an efficient allocation of testing resources [20] and prioritization of efforts [17].

3. The identification of fault-prone modules may trigger more thorough design of risky components, thus improving software architecture [20].

4. The prediction of fault count data helps predicting the quality to be achieved from a software [10].

5. “A good defect prediction model is an important first step towards pricing maintenance contracts, estimating support costs such as maintenance staffing, and creating software insurance” [21].

6. Software reliability prediction in terms of prediction of faults is indispensable with respect to determining optimal time to stop testing.

In short, accurate predictions are helpful for “…tendering bids, monitoring progress, scheduling resources and evaluating risk factors” [21].

Keeping in view the above benefits, there are several predictive models proposed in software engineering literature. These models range from traditional statistical (regression) models to machine learning models and models making use of both traditional and machine learning techniques. Despite the presence of these many models, there has not been consensus in the research community regarding which approach is the most suitable one. “Indeed, very contradictory results have been reported in studies comparing an arbitrary function approximator (or machine learning model) with a function. Furthermore, the performance of arbitrary function approximators varies widely across studies” [26].

The most recent study that we know of, related to fault predictions, is by Lessmann et al. [20], who established that metric-based classification is useful. But still the authors suggest more research to improve convergence across studies.

There has been considerable interest in understanding the reasons behind contradicting results in software engineering prediction studies. Several studies have identified various reasons attributed to diverging results of software engineering prediction studies [26, 20, 16, 9]. A brief summary of such reasons is given below.

Nature of software engineering data sets. Software engineering data sets have properties that challenge effective analysis and modeling. These properties include missing data, presence of many explanatory variables (both continuous and discrete), complex interdependencies and collinearity between the variables, heteroscedasticity, presence of outliers (or atypical variable values) and small size of data [4, 29, 30]. Although some of these issues can be reduced to some extent (discussed later in Section 4), others cannot be and thus any model derived using such data would be less reliable.

Incomplete understanding of software development process. Since software data is complex and have many variables, we possess a poor understanding of the software development process. Therefore, “…it is very difficult to make valid assumptions about the form of the functional relationship between the variables” [4]. Since variables selection is an important part of experiment planning, any short-
coming at this step is expected to bias the experimental results.

Misleading accuracy indicators. The software engineering research community has realized that there are certain accuracy indicators that are not only invalid but also unreliable. There are studies indicating that there is no consensus among use of various accuracy indicators and the choice of indicator determines the preferred prediction system which is of course not desirable [19, 25, 29].

4 Lessons learned and guidelines

In this section, we summarize our experiences and present useful lessons with respect to the different steps in the typical software engineering experimental process (Section 2) and the predictive modeling studies.

4.1 Definition

This step defines the basic purpose of the experiment and guides rest of the experimentation process. We found the use of the goal definition template [35] as a useful first step in experimenting with prediction models because it takes into account important aspects of objects, purpose, quality focus and perspective. We summarize one of our studies [3] in the goal definition template:

“Analyze the traditional and genetic programming (GP) techniques [objects] for the purpose of evaluation [purpose] with respect to model validity, goodness of fit and model bias [quality focus] from the point of view of the researcher [perspective] in the context of fault count data from three industrial projects [context]”

We found the definition step to be as important in experimenting with predictive models as generally in software engineering experimentation.

4.1.1 Useful lessons

1. The objects define the scope of the experiment [35], therefore a background in related literature helps to clarify the need of an experiment, such as the expected contribution of the experiment in increasing our understanding of the trade-offs among different prediction systems.

2. The definition step is expected to be refined and revised before experiment operation due to a gradual increase in problem understanding during planning.

4.2 Planning

Once the need of an experiment is identified, it is important to properly plan the experiment. Context selection [35] is one of the steps in experiment planning. With respect to context, we were engaged in performing experiments in an on-line situation, which in our case, meant that the experiments were based on the data collected from real-world projects. On the other hand, other context are also possible, like making use of replication and simulation.

Simulation can be a possible way out of the problem of having limited data. Pickard et al. [28] were the first to propose the use of simulation for evaluating software models [30]. Simulation can be used in different situations, i.e. a large number of values can be created that follow a particular data distribution e.g. Gaussian. It is also lot easier to compare a true distribution of data against another one and it enables study of complex phenomenon where analytical solutions are difficult to reach [13]. Shepperd et al. [29] also recommend simulation, as a way to counter difficulties when collecting large industrial data sets.

We also consider replication of studies as an important research methodology to better understand and generalize the study results. A replicated study helps to identify any anomaly or similarity among study results and an insight into the factors causing it. Especially with contradicting results in prediction studies in software engineering, replication would contribute to possibly present different perspectives on the problem for an increased understanding.

Within selecting context for an experiment, it is often required to select models or methods for comparison. Within prediction studies in software engineering, it is important to select a representative set of models for comparisons. Having a representative set of comparative models would increase generalizability of results and conclusion validity. There can be several motivations of selecting comparative models which are largely derived by the research hypothesis. As an example, in one of our studies [3], three fault count models were selected for comparison as the goal was to compare a family of fault count models. Also, these models presented a fair representation in terms of different forms of the growth curve.

After context selection, a hypothesis is formally stated which is later validated using statistical tests. We formulated the following null and alternative hypotheses in [3]:

\( H_{0_{-gof}} \): The GP evolved model does not give significantly higher goodness of fit as compared with traditional models.

\( H_{1_{-gof}} \): The GP evolved model gives significantly higher goodness of fit as compared with traditional models.

We also need to select the independent and dependent variables. Independent variables are those that are manipulated and controlled; and their effect is measured in depen-
dent variables [35]. This is an important step in experimental planning, especially in metric-based regression models.

The selection of subjects [35] is also an important step in experimental planning. In prediction studies, subjects might vary; e.g., in one of our studies [3], we used weekly fault count data sets collected during the testing of three large scale software projects as the subjects. Since selection of subjects is related to the ability to generalize the results [35], we are not certain in stating what is a reasonable level of generalizability. We plan to investigate this aspect further in future studies.

The planning of an experiment also include selecting an experimental design that is suitable for using statistical analysis [35]. We used one factor with two treatments in one of our studies [3] which compared fault prediction using GP with three traditional fault prediction models. So in this case, factor is the fault prediction method and the treatments are use of GP and traditional models. As part of design, the hypothesis should be analyzed to select appropriate statistical analysis method [35]. In the formulated hypotheses in our study ($H_{0-gof}$, $H_{1-gof}$), it is evident that we need to use a goodness of fit test for testing the stated hypotheses.

It is also important to select instrumentation (objects, guidelines and measurement instruments) [35] for the experiment. The object in the case of prediction studies in software engineering might be fault data sets, while there needs to be some preparation (guidelines) if new methods are to be experimented. If there is a need for data collection, measurement instruments need to be developed, e.g., use of forms [35]. In building prediction systems, data collection might be in the form of model outputs on validation portion of the data set.

The last step with experiment planning is validity evaluation which is discussed in much detail in [35] along with the validity threats for conclusion, internal, construct and external validity. We realized that these validity threats are equally applicable to prediction studies in software engineering as in experimental software engineering in general.

### 4.2.1 Useful lessons

1. Data sets required for experimentation are difficult to get due to several reasons, e.g., data being confidential and lack of enough information to be extracted from the data. Simulation can be used to generate data that follows a particular distribution and may help reaching valid conclusions [28, 13, 29]. It is also possible to get publicly available data sets, e.g., PROMISE data sets [2] and from NASA IV&V Tools Lab [1].

2. Some considerations in selecting models for comparison are helpful:
   - (a) Availability of software implementing the model algorithms.
   - (b) Active research in a particular modeling mechanism (as in [30]).
   - (c) Specific data requirements of a particular model, e.g., Shooman’s exponential model’s hazard function requires knowing the parameters of total number of instructions in the program and debugging time since the start of system integration.

3. The stated hypotheses need to be specific so that it can either be refuted or accepted using statistical tests.

4. It is also important to understand the process variables and their alignment with the context in focus so as to identify the correct experimental factors.

5. The decisions regarding the selection of variables, their scale type, stated hypotheses and types of statistical tests to perform are not independent but inter-related.

### 4.3 Operation

In the operational step, the subjects are exposed to treatments [35]. The operation step consists of three further steps of preparation, execution and data validation [35]. In the context of experimental software engineering in general, the preparation step would mean dealing effectively with human subjects on most of the occasions. On the other hand, it is not necessary in prediction studies in software engineering to involve human subjects. Thus the preparation step here would resemble the instrumentation step in the planning phase of experimental software engineering which includes choosing objects, guidelines and measurement instruments.

As part of the preparation steps, especially in prediction studies in software engineering, it might be a reasonable approach to do data preprocessing. Within machine learning techniques, data preprocessing may lead to improved results. The preprocessing of data using attribute selection, attribute discretization, data transformation and data cleansing [34] might increase chances of improved results. Attribute selection works to eliminate irrelevant attributes and can be done both manually and automatically. Attribute discretization is a kind of data transformation that involves converting numeric attributes into small number of distinct ranges [34] to make them suitable for some classification algorithms and finally, data cleansing refers to various ways to make data noise-free. Therefore, it seems useful to analyze the data sets to avail opportunities to make the data more suitable for different techniques. For example, factor analysis can be used to remove multicolinearity which causes incorrect statistical tests and misleading coefficient signs.
In the execution step of experiment operation, a typical software engineering experiment would collect experimental data to be used for statistical analysis. But as mentioned earlier, the execution step of experiments in predictive models mostly deals with model outputs on validation part of the data set. This also serves the purpose of data validation which is the third step in the experimental operation [35].

It is common in research on prediction models to divide the data set into training (or fix) and test set. To increase expectancy of unbiased results, an impartial data splitting or cross-validation technique is desirable. There are typically two types of cross-validation, $n$-fold (leave-one-out) and $v$-fold [26], where $n$ is the number of instances in the data set and $v$ is some number smaller than $n$. In $n$-fold cross-validation, the learning method is trained on all, except one, instances. The model’s correctness is then evaluated on the remaining instance. In this way the results for all $n$ members of the data set are averaged to present a final error estimate [34]. In $v$-fold cross-validation, the data set is split in to $v$ partitions and each partition in turn is used for testing and the remainder is used for training. Standard practice is to use $v = 10$ for a 10-fold cross-validation. There is still no consensus on which cross-validation method is the most suitable. Myrtev et al. [26] describes $n$-fold as being more practically suited to real-world software development situations than $v$-fold, but on the other hand, $v$-fold is less computationally intensive. However, any form of cross-validation will increase the transparency and unbiasedness of model results.

### 4.3.1 Useful lessons

1. Preprocessing of data might help a machine learning algorithm in converging to a suitable solution.

2. It is important to keep the training and test sets as independent because the validation of the learned model on an independent test set is expected to closely match the fresh data that will be applied in practice [34].

3. Within machine learning and evolutionary computation approaches to prediction studies, it is expected that some experimentation at the operational step might be required for adjusting the algorithmic parameters.

4. For machine learning approaches to predictive models, it is especially important to document the algorithmic settings and control parameters during the operation so as to serve as a basis for reaching optimal tuning of these parameters in future comparative studies.

### 4.4 Analysis and interpretation

Once data is collected during experiment operation, analysis and interpretation step can begin. Analysis and interpretation is done in three steps: descriptive statistics, reducing the data set and hypothesis testing [35].

Descriptive statistics are used for knowing the data distribution. Graphical visualization in the form of scatter plots, box plots and histograms illustrates the properties of data sets [35]. Therefore, descriptive statistics represent useful tools for data exploration.

Data set reduction deals with detecting outliers; while hypothesis testing makes use of parametric and non-parametric methods to test the formulated hypotheses.

Earlier studies on predictive accuracy of competing models did not use to test results for statistical significance and drew conclusions without reporting significance levels. This is, however, now less practiced as more and more studies report statistical tests of significance, e.g. in [15] one-way ANOVA and Tuckey’s multiple comparison tests were used to analyze the predictive performances of the different methods with respect to the absolute average error (AAE) and absolute relative error (ARE) values. Statistical tests of significance are important since it is not reliable to draw conclusions merely on observed differences in means or medians because the differences could have been caused by chance alone [25]. The use of statistical tests of significance comes with its own share of challenges about which tests are suitable for a given problem. A study by Demšar [8] recommends non-parametric tests for statistical comparisons of classifiers; while elsewhere in [5] parametric techniques are seen as robust to limited violations in assumptions and as more powerful (in terms of sensitivity to detect significant outcomes) than non-parametric.

As we discussed earlier in Section 3, there is no consensus with regards as to which accuracy indicator is the most suitable for the problem at hand. Commonly used indicators suffer from different limitations, for details see [13, 29]. One intuitive way out of this dilemma is to employ more than one accuracy indicator, so as to better reflect on a model’s predictive performance in light of different limitations of each accuracy indicator. This way the results can be better assessed with respect to each accuracy indicator and we can better reflect on a particular model’s reliability and validity. However, reporting several measures that are all based on a basic measure like mean relative error (MRE) would not be useful [13]. In [27], measures for the following characteristics are proposed: Goodness of fit (Kolmogorov-Smirnov test), Model bias (U-plot), Model bias trend (Y-plot) and Short-term predictability (Frequent likelihood). These measures, although providing a thorough evaluation of a model’s predictions, lacks a suitable measure for variable-term predictability. In [15, 24], average relative error is used as a measure of variable term predictability. To our knowledge, we are not aware of any critique of such an approach for variable-term predictability.
As an example of applying multiple measures, one of our recent studies [3] used measures of prequential likelihood, Braun statistic and adjusted mean square error for evaluating model validity. Additionally we examined the distribution to residuals from each model to measure model bias. Lastly, the Kolmogorov-Smirnov test was applied for evaluating goodness of fit. More recently, analyzing distribution of residuals is proposed as an alternative measure [19, 29]. It has the convenience of applying significance tests and visualizing differences in absolute residuals of competing models using box plots.

We see examples of studies in which the authors use a two-prong evaluation strategy for comparing various modeling techniques. They include both quantitative evaluation and subjective qualitative criteria based evaluation because they consider using only empirical evaluation as an insufficient way to judge a model’s output accuracy. Qualitative criterion-based evaluation evaluates each method based on conceptual requirements [16]. One or more of these requirements might influence model selection. Examples of qualitative criteria include [6, 16, 22, 23]:

1. Does the model require specification of the form of relationship between the variables or does it determine its own structure?
2. How robust is the model in dealing with outliers (insensitivity to noise)?
3. Is the model’s output affected by small data sets, and if yes, how much?
4. Can the model adjust to incorporate additional data or does the model require regeneration on the combined data set?
5. Is the process of model building transparent and reasoning process visible?
6. Is the model able to capture complex relationships in data?
7. Is the model capable of including known facts to improve and refine its output?
8. How easy is it to configure the technique used for modeling (ease of configuration)?
9. What time and memory resources are required for model building?
10. What is the extent of generality of model results for diverse data sets?
11. What is the applicability of the model in different life-cycle phases?

We can assign subjective ratings to above mentioned criteria to better explain the trade-offs in selecting a particular model. We argue that both quantitative and qualitative factors play an important part to establish the validity of a model. Therefore it should be possible to give a definite structure to these concepts to develop a multidimensional model evaluation system which gives proportionate weighting to empirical and qualitative factors for model assessment.

4.4.1 Useful lessons

1. It is important to satisfy the assumptions of statistical tests before considering to apply them in practice. According to Fenton et al. [9], one of the main reasons of invalid conclusions in empirical studies is not satisfying the assumptions of a statistical technique. One of the important assumptions of using appropriate statistics is the scale type of measure. As pointed out in [5], it is not always easy to figure out the scale type of measures in software engineering. This complicates the decision of applying parametric vs. non-parametric tests; but Briand et al. [5] demonstrates that if a researcher is confident that the scale type is between ordinal and interval, then it could be treated as being on an interval scale because commonly used parametric tests are robust to non-linear (not exponential) transformations of the interval scale.

2. Another decision to take is to select a suitable significance level ($\alpha$) for hypothesis testing, though the most commonly used significance levels in software engineering are 0.01 and 0.05. The significance level shows the probability of committing a Type I error (incorrect rejection of null hypothesis). The test is more sensitive if lower values of $\alpha$ are chosen and chances of finding significant results are more. Also lower values of $\alpha$ requires less magnitude of effect size, which is important with respect to software engineering where collecting large data sets is not always possible.

3. Using multiple accuracy indicators, that measure the same property, might help increase the conclusion validity of an experimental study.

4. Analysis of differences in absolute residuals of competing models is a useful way of comparison, allowing an experimenter to check for model bias as well as to apply significance tests.

5. Use of qualitative criteria based evaluation is a useful way to complement the quantitative evaluation of model outputs.
4.5 Presentation and package

The final step in the process of experimental software engineering is to decide upon the experiment report outline [35]. Wohlin et al [35] provides with a generic structure for such a report, containing an introduction; problem statement; experiment planning; experiment operation; data analysis; data analysis and interpretation. This structure facilitates reporting both general software engineering experiments and also prediction studies in software engineering.

4.5.1 Useful lessons

1. The reporting structure, if followed with all its constituents, supports replication which is one of the important aims of experimental software engineering.

5 Discussion

Evaluation of software engineering prediction systems makes up an important field in experimental software engineering. In part, issues faced by software engineering prediction systems are similar to the broader issues in experimental software engineering. We often encounter noisy and incomplete data, definition of appropriate measures is difficult and there are trade-offs in applying statistical analysis. There are alternatives to select at different stages of the experimentation process when evaluating prediction systems. With so many factors that might influence the credibility of prediction results, it is sensible to make use of best practices and recommendations at various stages of the experimentation process.

We also observe that data set characteristics have a significant impact on getting results with a particular prediction system. Shepperd et al. [30] showed that step-wise regression produced the most accurate predictions for normal and normal+outlier data sets, and machine learning techniques showed better predictive performance on data sets having collinearity, outliers and normal distribution. Also the data splitting scheme into training and test sets and size of training set had significant impact on a model outcome. This supports using cross-validation and preprocessing wherever feasible.

The inherent problems in software engineering data sets have encouraged non-traditional modeling mechanisms but each one of them come with inherent limitations and require further experimentation. Various alternative models offer a choice to the end user in selecting the most appropriate alternative, especially when there is no significant trend in accuracy prediction of a particular model [7].

More and more studies on software fault predictions are making use of an analytical approach that complements the statistical evaluation. There is a realization that “...statistics on its own does not provide scientific explanations” [9].

We believe that it is useful to investigate the modeling of long term behavior, which would validate the true potential of a model on practical grounds. Also, models should ideally be validated to a wide range of commercial software systems (e.g. operating systems, servers, web browsers) [21] as they represent suitable variations in their respective operational profiles. Moreover, there is a need to design new software metrics that incorporate both quantitative and qualitative criteria. Another potential area of future work includes investigating the impacts of cross-validation method chosen on the predictive model’s performance.

6 Conclusion

The problems faced by prediction studies in software engineering are not new but interestingly still pose threats to the validity of these studies. One way to move towards better convergence of study results is to follow a process-oriented approach. We found that the basic steps of experimentation in software engineering are relevant to prediction studies; while there are issues specific to each step in the experimentation process which might require more attention with respect to prediction studies.

References


Software Error Impact and Cost Analysis

Ralf Gitzel, Simone Krug, Martin Schader
Software Error Impact and Cost Analysis

Ralf Gitzel
ABB Corporate Research Center, Germany
Ralf.Gitzel@de.abb.com

Simone Krug
University of Mannheim, Germany
skrug@wifo.uni-mannheim.de

Martin Schader
University of Mannheim, Germany
martin.schader@uni-mannheim.de

Abstract
Software delivered to a customer usually is not free of defects. Many software providers are aware of the cost resulting from the need to fix these defects. On the other hand, the estimation of costs arising for the end user has been neglected so far. While this impact might be negligible for many applications (e.g., text editors), industrial software applications or administrative software such as ERP systems support business-critical processes. The cost of system downtime alone is an important factor for software producers.

In order to assess the monetary effects of a software error, this project includes empirical research targeting the detection of the links between the cause of an error, its importance to the functionality of the software product, and the impact on the costs of the user.

Our goal is to propose an error impact model that provides a conclusive approach which can be applied by practitioners and extended by researchers.

I Introduction
IT is an important cost driver in enterprises. Over the past years, hardware has become a commodity, so that the research focus has more and more shifted towards the optimization of software costs. So far, research has centered on the supplier, i.e., the cost for software development, version management, etc. However, there is another side to the coin – the cost incurred on the customer’s side.

Our focus of research is on the costs caused by software errors that affect the end user. While in many cases the costs to the user is minimal, there are scenarios where faults in software can be the cause of significant costs. This is particularly true for software supporting critical business processes. For example, defects in industrial software can lead to production downtimes (where a rate of 50,000 $ per hour is not unusual), reduced product quality, or suboptimal performance.

Even the fixing of problems on the client side (typically through patches) can incur high costs. An update will at least trigger a service case and might even result in significant system unavailability. More exotic cases are the costs for re-certifying a pharmaceutical plant after applying a patch that caused minor changes to the basic plant functionality.

In this paper, we propose a research project that will address these issues. The paper identifies the key challenges and proposes the adaptation of approaches taken from the domain of hardware. The discussion will follow the following structure. In the next section, a brief overview of the problem is given. Section 3 will then describe existing techniques for assessing the consequences of error for the end user taken from different domains. Our proposed research concepts are shown in section 4. We end the paper with a short conclusion.

II Impact of Software Errors on the End User
The impact of software errors on the end user is not trivial to classify. One reason is that there are errors and there are failures but the two are not equivalent. The term error describes the state of a component which does not comply with its specifications, whereas a failure is characterized by the malfunctioning of certain functionality. Thus, an error marks the origin of a failure, but not all errors inevitably lead to a failure. Therefore, in order to identify the impact of errors on the end user, the chain of causality between errors, failures, and the effect on the business process has to be identified (cf. figure 1).

Figure 1: Impact of error on the business process
The view cannot be limited to either end of the chain. On the one side, the impact on the business process...
influences the amount of cost. For example, the well-known Pentium bug was problematic for computationally intensive tasks that relied on certain calculations but had little consequence for text editing. Similarly, if the connection between error and failure is not understood, we cannot influence the costs to the user as we cannot directly influence the failures.

As we will discuss in the next section, there are already a number of tools that either look at parts of the causal chain or look at similar systems in related (i.e., non-software) domains. We plan to incorporate these partial solutions into a holistic approach that provides much needed support to both software customers looking to estimate costs and software vendors keen on improving the perceived quality of their products.

III Existing Tools for Error and Fault Analysis

As we have already mentioned, there are a couple of existing approaches to the problem described in the previous section. We will describe each in turn and show their usefulness to our project in the next section.

III.1 FMEA

FMEA (Failure Mode and Effects and Analysis) is a traditional reliability and safety analysis technique, which originates from engineering science. Its idea is to detect possible weaknesses in the early stages of development and hence contribute to error prevention and augmented quality of the end product. Kennedy [6] sees FMEA as “a useful tool to identify any potential design and process related failure modes” and lists additional purposes:

- to determine the effects of the failure modes.
- to determine the root cause of the failure modes.
- to prioritize actions by using a ranking system for the failure mode effects in terms of probability of occurrence of the failure mode, severity of the effect of the failure mode, and probability of detection of the failure mode through manufacturing.
- to identify, implement, and document corrective actions to address failure modes with rankings that are considered unacceptable.

We can illustrate the concept of FMEA using hardware parts of a process automation network as an example. The main network in the plant connects the controllers (real-time computers directly affecting production) with the computers of the human-machine-interface (HMI). The HMI includes operator stations for manually controlling the process and engineering stations for programming the controllers.

![Figure 2: The main network](image)

The left side of figure 2 shows a schematic presentation of the main network, which consists of four switches and a redundancy manager (shown in dark) organized in a ring-like fashion. The arrows denote the preconfigured route of packages from machine 1 to machine 2. The redundancy manager works like a regular switch under normal circumstances. However, if one of the switches breaks (right side), the redundancy manager will detect the damage and organize a rerouting using the other direction of the ring. Since all important network nodes are connected to two different switches, a rerouted system will still allow all nodes to be reached.

While this type of system has a myriad of failure possibilities, we will focus on only a few (specified in table 1) to keep the example manageable and interesting.

<table>
<thead>
<tr>
<th>Possible Failures in the Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Switch logic becomes flawed, routing packages to the wrong addresses</td>
</tr>
<tr>
<td>- The switch’s connector to the network breaks</td>
</tr>
<tr>
<td>- The network card of a computer malfunctions</td>
</tr>
<tr>
<td>- The switch functionality of the redundancy manager fails</td>
</tr>
<tr>
<td>- The rerouting functionality of the redundancy manager fails</td>
</tr>
</tbody>
</table>

Table 1: Possible network failure scenarios

The following FMEA table (see table 2) assumes a system without a redundancy manager. The first column describes a component, the second column one of its functions and the third an operational mode. In our case, there are only two modes, operational and switched off, with the latter having no failure effects.

The column ‘Failure Type’ describes different possible failure effects with local and system-wide consequences. So far, only a verbal description has been given, which has to be translated into quantitative expressions. Thus, the last columns (in gray) give an estimate for the probability of occurrence, the impact on the system and the risk of non-detection. The latter is important as problems that cannot be detected will have greater impact on the system output as the mistake cannot be corrected. By multiplying the three values to get a priority, it is...
possible to identify problems that have to be addressed. For example, anything with a priority above 100 might be considered problematic.

The assessment of the errors in table 2 is for illustrational purposes only but helps showing the benefit of using FMEA. The middle line describes the scenario of connector breakage. Since there are several switches involved, the probability of failure is considered to be relatively high (a value of 5). The impact on the system is deemed very high.

After the mitigation through the redundancy manager (see table 3), the risk is considered to be reduced. A failure no longer has a fatal impact on the system. The additional failure modes introduced through the new component are lower in risk priority and do not require further action.

<table>
<thead>
<tr>
<th>Component</th>
<th>Function</th>
<th>Operational Mode</th>
<th>Failure Type</th>
<th>Local Failure Consequence</th>
<th>System Failure Consequence</th>
<th>Probability (1-10)</th>
<th>Impact (1-10)</th>
<th>Risk of Non-Detection (1-10)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Signal Routing</td>
<td>Operational</td>
<td>Wrong Routing</td>
<td>Data Package sent to wrong neighboring switch</td>
<td>Package takes longer to reach target</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Signal Routing</td>
<td>Operational</td>
<td>Connector broken</td>
<td>No signal leaves router</td>
<td>Signal is lost until a reroute can be triggered</td>
<td></td>
<td>5</td>
<td>10</td>
<td>3</td>
<td>150</td>
</tr>
<tr>
<td>Computer</td>
<td>Signal Encoding</td>
<td>Operational</td>
<td>Network Card Broken</td>
<td>No Signal Generated</td>
<td>No communication possible</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 2: FMEA without redundancy manager

<table>
<thead>
<tr>
<th>Component</th>
<th>Function</th>
<th>Operational Mode</th>
<th>Failure Type</th>
<th>Local Failure Consequence</th>
<th>System Failure Consequence</th>
<th>Probability (1-10)</th>
<th>Impact (1-10)</th>
<th>Risk of Non-Detection (1-10)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Signal Routing</td>
<td>Operational</td>
<td>Wrong Routing</td>
<td>Data Package sent to wrong neighboring switch</td>
<td>Package takes longer to reach target</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Signal Routing</td>
<td>Operational</td>
<td>Connector broken</td>
<td>No signal leaves router</td>
<td>Signal is lost until a reroute can be triggered</td>
<td></td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>75</td>
</tr>
<tr>
<td>Computer</td>
<td>Signal Encoding</td>
<td>Operational</td>
<td>Network Card Broken</td>
<td>No Signal Generated</td>
<td>No communication possible</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>Redundancy Manager</td>
<td>Signal Routing</td>
<td>Operational</td>
<td>Connector broken</td>
<td>No signal leaves router</td>
<td>Signal is lost until a reroute can be triggered</td>
<td>2</td>
<td>10</td>
<td>3</td>
<td>60</td>
</tr>
<tr>
<td>Rerouting</td>
<td>Periodical System Check</td>
<td>Manager Fails to Recognize Problem</td>
<td>No Rerouting Occurs</td>
<td>A switch breakdown will not be mitigated</td>
<td></td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>42</td>
</tr>
</tbody>
</table>

Table 3: FMEA with redundancy manager
III.2 Software FMEA

FMEA has already been extended to the needs of software engineers, in form of the software FMEA [3]. Software FMEA is different from the FMEA employed for mechanical or electrical systems. While failure modes of software are generally unknown, the reasons for failure of component like relays or resistors can be identified and tested, such as for example wear-out or aging.

It is also important to recognize that there is a difference between hardware failure rate and software failure rate. While failure rates of hardware tend to decrease at the beginning and increase again at the end of the life cycle, the software failure rate in theory decreases continually [11]. This, of course, refers to software systems, where no new defects are introduced, whereas in real-world systems it has to be considered that changes bear the potential of introducing new errors to the code, even if their purpose was to fix a defect. Recently, research has been conducted [7] on how to decide whether a change will introduce new defects or not. In general, regarding the multiple changes introduced to a system in its life cycle, one would expect there to be less severe defects and a decrease of the overall failure rate.

Many works has been published on the improvement of the analysis of cost of maintenance of software [1]. But so far, software FMEA does not include any reference to the costs for the end user.

III.3 Fault Trees

It is also important to realize that failures are interconnected and, especially in complex systems, the combination of multiple minor errors can lead to far-reaching failures within the system [5]. This interconnection between failures can be visualized by fault trees [13].

Revisiting again the example case, we can see the additional information conveyed by a fault tree. Figure 3 shows all cases from table 3 that can lead to a breakdown of network communication. As can be seen on the right side, the failure of a single network card will make parts of the network inaccessible. For an internal failure of the network, at least one switch’s connectors has to fail in conjunction with the rerouting system of the redundancy manager in order to cause the network to go down. This interrelation is shown on the right side of the figure.

It is interesting to see that with the redundancy manager, a new type of failure has been introduced that can lead to a system failure by itself. If the redundancy manager’s connector fails the network fails. As a logical consequence, the redundancy manager is a useful addition only if the probability of this event is very low.

III.4 Software metrics

Software metrics can be utilized in order to arrive at an estimate of the number of errors to be expected in a software system. Luke [8] found that the defect rate is positively linearly correlated to the complexity of a program. The most widely-used methods of measuring program complexity are counting the number of instructions in a program [10] or analyzing the coherence of different functions. One of the best known approaches is the set of metrics introduced by Halstead [4]. His approach to program complexity is static, meaning that the analysis is based on the program’s code rather than its execution. It takes into account the number of distinct operators and operands in a program, as well as their number of actual occurrences in the source code:
The size of vocabulary is \( \eta \): The number of distinct operators appearing in that implementation, e.g., \(+\), \(-\), \(>\), if.

\( \eta_2 \): The number of distinct operands appearing in a program, e.g., variables, constants.

\( N_1 \): The total number of occurrence of operators in a program.

\( N_2 \): The total number of occurrence of operands in a program.

The size of vocabulary is \( \eta = \eta_1 + \eta_2 \). As the program consists of the sum of all operators’ occurrences and all operands’ occurrences, the implementation length is \( N = N_1 + N_2 \). Halstead defines the program volume as \( V = N \log_2 \eta \) and gives \( B = V / 3000 \) as an estimation for the number of errors in the program. Halstead also provides metrics for the program level and the implementation effort, allowing estimating the required manpower for the development of a software system.

The Halstead complexity measures are useful for comparing different algorithms [10], which is especially interesting as it is thereby possible to analyze programs independently of the programming language. An additional advantage is that these measures can easily be automated. What can not be delivered are insights on the complexity of the program flow. Other approaches, such as, for example, McCabe’s cyclomatic number [9], try to overcome this shortcoming.

There are also other alternatives to consider such as the metrics suite proposed by Chidamber and Kemer [2]. Their object-oriented metrics reflect complexity and testability of a program. For example, the Coupling between Object Classes (CBO) gives an indication as to how difficult it will be to run unit tests.

\[ \begin{align*}
\eta_1 & : \text{The number of distinct operators appearing in that implementation, e.g., }+, -, >, \text{if.} \\
\eta_2 & : \text{The number of distinct operands appearing in a program, e.g., variables, constants.} \\
N_1 & : \text{The total number of occurrence of operators in a program.} \\
N_2 & : \text{The total number of occurrence of operands in a program.} \\
V & = N \log_2 \eta & \text{program volume} \\
B & = V / 3000 & \text{estimation for the number of errors in the program.}
\end{align*} \]

IV Proposed Approach

Changes to software systems can be divided into two categories, defect repairs (“bug fixes”) and the addition of new features. Regarding all changes, most studies find that there are more of the first kind than of the latter; e.g., Stark [12] found that 62% are fixes and 38% are modifications. In the first stage of our project the focus will lie on the first aspect, defect repair. Since updates also may contain new defects, this also has to be taken into account in the progression of the project.

We propose to establish a model for the analysis of software error impact and induced costs for the customer. In order to create the error impact model, the following steps have to be accomplished in order to capture all aspects mentioned in figure 1:

- Select one (or more) software metric to establish a relationship between the source code and the expected defect rate. The focus here lies on the impact of a defect to other parts of the software.

- Allocate a failure to costs to span the gap between the estimated impact of an error and its consequence for the end user.

Current research activities attend mainly to the first step and include the close monitoring of errors occurring in a software product. Central attention is here devoted to the collection of information about the cause as well as the effects of an error. This initial data collection will help to adapt the proposed model, if needed. Our planned research includes the examination of at least two different software products, one in the public sector and one in process automation industry.

In section III we have discussed several tools for the assessment of failures. Each of these holds a different potential as we will discuss in the remainder of this section.

FMEA is an established tool for the analysis of failure modes in hardware systems. As illustrated by the proposal of a software version, the basic concept is attractive for the software world as well. Its focus on design problems suits the design-intensive aspect of software quite well. However, an FMEA is only a preliminary step in the identification of problems and, in particular, it lacks a precise view on the combined effects of multiple errors.

The software FMEA includes the description of the measures to be taken in order to prevent or mitigate the consequences of software failures. These measures can also be prioritized, as it is possible to calculate the probability of the occurrence of the failure mode [6].

Fault trees introduce the additional aspect of relationships between problems in a system. Given the modular nature of well-designed software, it should be possible to apply fault trees to software. This way, we gain deeper insight into the causal connections within the system.

Given an identification of problems, their outcomes and their interaction, there remains the need to identify probabilities of problem occurrence in order to quantitatively estimate the monetary effects. The aforementioned software metrics could serve as a good basis for a simulation-based calculation of costs. However, it is crucial to identify metrics well-supported by empirical evidence. Also, the employment of metrics to estimate the number of expected errors is not without controversy. The problem is that they often provide nothing more than an abstract number, e.g., representing the number of expected defects. What they are not able to provide is an understanding of the consequences. The challenge will be to span the gap from the estimations to real-world systems.

V Conclusion

In this paper, we have briefly discussed motivations for a software cost estimation based on a customer’s
view. Based on this motivation, we have shown several elements that we consider useful for this purpose and have sketched out how they would add up to a holistic view.

Given the tools for such cost estimation, an end user will be supported in his decision as to which software to invest in. The cost induced by software errors within the scope of the customer is also relevant to software providers, as they wish to distinguish themselves from their competitors. In consequence, software products become more comparable, as more customers demand estimation of error impact.

References


Effect of Functional Similarity for Establishing Relation between Effort and Functional Size

Seckin Tunalilar, Onur Demirors
Effect of Functional Similarity for Establishing Relation between Effort and Functional Size

Seçkin TUNALILAR
ASELSAN MGEÖ Division
stunalilar@mgeo.aselsan.com.tr

Onur DEMİRÖRS
Informatics Institute
demirors@ii.metu.edu.tr

Abstract

Although many models have been proposed to build a relationship between effort and size, we still do have difficulties for effort estimation. In this study we present a case study on one of the aspects of functional size estimation: Functional Similarity (FS). We investigated whether functional similarity has effects on software sizing and productivity calculations. For the case study, projects from different application domains were selected and CosmicFFP method has been applied to all. We also investigated the effect of FS on different phases.

1. Introduction

Effort estimation is a critical activity for project management since planning, scheduling and budget depends on the estimated effort. In most cases effort required to build a software product is assumed to be a function of size of the product. Therefore, variations in size are those that determine the variations in the related effort and cost. All proposed effort models proved themselves using their own limited data-sets but none of them has been widely accepted by software society as a commonly used model.

Although function point (FP) methods and their derivatives may be accepted as valuable tools for software sizing, there are some unaccounted issues related to structure of the software product. In this study we mainly concerned one of these important aspects; Functional Similarity (FS).

Most software products have functionally similar modules that can be identified by using functional user requirements. An analyst may detect some of these modules by further investigation of similarly stated requirements. However, it is not always easy to recognize all functionally similar software modules at the beginning of large projects.

In FP size calculations, software requirements are mapped to functional processes. These processes are made up of functional sub-components. A way of finding similarities among functional processes would be comparing sub-components of functional processes. If they are similar, then we can assume these processes are functionally similar. Regardless of the aimed logical functionality, if functional processes of modules are highly similar then the effort for developing that software might not be proportional to the functional size. Several studies examined the concept of functional similarity[11] [14] for distinguishing similarities and quantifying reuse potential of projects. However validation of these results for different application domains and how to reflect these results into effort models has not been studied. In this paper our motivation is to observe how functional similarity has an impact on different application domains and phases of projects.

This paper is organized as follows. Section 2 presents a literature survey about the recommended solutions on size and effort relationship. Section 3 presents the case study, section 4 summarize our results.
2. Related Research

Although we have well defined guidelines to measure the functional size of software products, there are still difficulties in measuring the functional size that directly affect the magnitude of effort required to develop software products. A number of studies are performed to find a way to build this relation reliably. We present a summary of these studies from three different points of views:

- Studies related to structure of the measured product such as Algorithm complexity, Functional Similarity.
- Studies on new sizing methods
- Studies on building reliable data-sets.

While sizing functionalities we measure the size of software from user’s point of view. Cosmic FFP expand this view to include developer’s point of view. However, the structure of the product is not considered as part of the size measurement methodologies. From the developer’s view point creating a functionally similar module to obtain new logical feature lowers the effort required. On the other hand, increasing algorithm complexity may require considerably larger development effort per functionality. Without a doubt finding an agreeable way of integrating them to existing size measurement methods may result in better prediction of software effort.

Functional similarity concept is first defined by Meli as the re-utilization of existent logical data structures and functionalities to build up new logical features [12]. But how the similarity should be considered for effort and size calculations is not defined in these studies. Santillo and Noce [14] defined the same concept, and suggested a model for calculating the size considering the similarities as “Worked FP model”. The model use predefined reuse adjustment coefficients for every reused-function. However the method is not validated by means of industrial experiments. Similar set of coefficients is also proposed by NESMA for enhancement projects [10].

Santillo and Abran in [11] proposed an approach for uncovering the functional similarities among functional processes by analyzing similarities of data movements and data manipulations of functional processes of the COSMIC FSM method. The approach has two stages. In the first stage, named as “first order evaluation”, functional processes are compared only from data movements’ point of view. Functional similarity percentages among functional processes are decided by comparing the data group and data movement relationships. In the second stage functional similarities are determined by considering both data movement and data manipulation action types. Both stages are applied to all functional processes to find internal reuse potential of the product. Although this study comprises a method sorting out functional similarities, it does not provide any model for building the relation of size and effort. Tunalilar, Top and Demirors applied different approaches to identify the effect of functional similarity on the development effort [13] and evaluated the applicability of the Santillo and Abran’s method. They pointed out that this approach does not easily scale up on large scale applications since the evaluation process is error prone and time consuming. Based on these concepts Top [16] proposed a method called SiRFuS which computes Similarity Reflective Functional Sizes to attain adjusted functional sizes.

While using COSMIC FSM method, the number of entries and the number of exists for an algorithm can be counted, however data manipulations on them could also affect the development effort. Redgate and Tichenor proposed to disassemble algorithms and formulas into functional components, and measure their size and complexity [6][7]. This method does not require any new counting rules or patches and promises to give users a more accurate view of application size. However these models are applicable to only limited size of algorithms. Wang and Shao [8] developed a cognitive functional
size model, which is independent from the language. Cognitive weights for basic control structures have been introduced to measure the complexity of logical structures of software. The model however is not utilized for effort prediction.

In parallel with the improvements in development paradigms, mapping functionality concept to object oriented concepts became popular. In [1] and [2] size estimation methods which map some OO concepts into FP concepts are suggested. Class Point [3] is proposed to estimate system-level size. This method combines OO measures, which take into account specific aspects of each class such as complexity. Similarly, UML points[4], combines Use Case Points and Class Points to provide software system size information. Although these building blocks of software may be a good way to deal with the problem, granularity of subcomponents still depends on software analyst’s decision. All these new works would be beneficial if effort prediction models based on these measures are developed and validated for a significant number of projects. Up to now a small number of studies exist and data-sets are not available. Formulas for these models must be obtained empirically and calibrated over time to improve the accuracy of estimates. Even worse, a new study [5] on class point approach resulted that; researchers couldn’t find any relation between class point size and effort.

Another difficulty for specifying an agreeable effort model is the lack of reliable data-sets. With the aid of repositories like ISBSG and the ESA datasets, companies can compare themselves with respect to their productivity with others. However, the advantage of the multi-company projects database is still under debate. Even assuming that there is enormous number of data for similar projects, there is insufficient data to construct and test a model for effort prediction. Mendes and Lokan [9] claimed that a minimum parameter set is more applicable if collected completely and appropriately. Other problem related to data-sets is that most of the variables in these sets had more than 40% of their values missing, therefore excluded from the analysis. In order to use these incomplete data sets, imputation methods are proposed [15].

3. Case Study

In our case study we investigated the functional similarity effect on size and effort relationship. We formulated three research questions.

1. Does effort and size relationship be affected by functional similarity?
2. Do all application types be affected by functional similarity?
3. What is the effect of Functional similarity on different phases of life-cycle?

The company, where we carried out the case study, is certified by ISO and AQAP standards. Main business of the company includes design and manufacture of software intensive systems. Diversity of product lines of company allows software developers to work in several different types of development environments. Effort data is collected as person-day basis including all the details of tasks to be performed for each software configuration item (SCI). Groups developing SCI’s are formed according to previous experiences. Each group has similar skill level. Software Requirements documents of each case are conformant with the IEEE Standard 830-1998. Each type of applications is developed with the same environment, using the same language and by the same team in the same organization. We selected 3 different application types:

1. GUI applications for the Data Driven Control Systems, (GUI): These SCI’s are simulators of some existing products. They are used to generate and send an artificial data in order to test another system or used to show the results of an externally connected system to check the accuracy of the data. They have one or more data-interfaces to be connected to other systems and GUI’s for user to change or create a new data, or presents the externally generated results.
2. Embedded Software applications for Hardware Support, (HW): These SCI’s are embedded device drivers developed for specifically designed hardware. All existing subcomponents of hardware are controlled with the aid of this software.

3. Real-time Embedded System Application Software (ES): These SCI’s are developed using RTOS development environments and also include the communication and control software, algorithm processing software etc. In our case study we didn’t include algorithm processing software SCI’s.

The first step for the case study was to determine the functional size of the projects by using the COSMIC method without considering similarity issues. By using a basic formula (Productivity \((P) = \frac{\text{Size}}{\text{Effort (man day)}}\)) and without using any regression technique we found that productivity values of the projects for similar teams has large variance. Productivity value changed between 30,6 and 0,35 as given in Table 1.

| Table 1 Productivity Rates \((P = \text{Cosmic Functional Size Unit (Cfsu) / man-day)}\) |
|---|---|---|---|---|---|
| GUI-1 | Size(CFP) | Effort | Initial Prod. | % Reuse | Cfsu- After FS | P After FS |
| GUI-2 | 1384 | 49,5 | 27,9 | 88 | 208 | 4,2 |
| GUI-3 | 412 | 59,5 | 6,92 | 11 | 374 | 6,28 |
| GUI-4 | 1438 | 47 | 30,6 | 81 | 213 | 4,53 |
| GUI-5 | 129 | 29 | 4,44 | 25 | 93 | 3,2 |
| GUI-6 | 76 | 31 | 2,45 | 14 | 59 | 1,9 |
| GUI-7 | 986 | 64,5 | 15,29 | 56 | 423 | 6,55 |
| GUI-8 | 1347 | 104 | 12,95 | 83 | 396 | 3,8 |
| GUI-9 | 136 | 12 | 11,3 | 86 | 13 | 1,08 |
| GUI-10 | 277 | 28 | 9,89 | 79 | 119 | 4,24 |
| GUI-11 | 193 | 28 | 6,89 | 57 | 95 | 3,39 |
| HW-1 | 419 | 98 | 4,27 | 10 | 402 | 4,1 |
| HW-2 | 1740 | 471 | 3,69 | 6 | 1620 | 3,43 |
| HW-3 | 660 | 253 | 2,6 | 0 | 660 | 2,6 |
| HW-4 | 421 | 133 | 3,16 | 0 | 421 | 3,16 |
| ES-1 | 2040 | 336 | 6,07 | 16 | 1634 | 4,86 |
| ES-2 | 246 | 696 | 0,35 | 6 | 194 | 0,27 |
| ES-3 | 362 | 139 | 2,6 | 4 | 311 | 2,23 |
| AVE. | 8,66 | 3,48 |
| AVE.DEV | 6,22 | 1,19 |
| VAR | 87,01 | 2,69 |

To determine the reuse percentage level of the projects Santillo and Abran’s Method first order evaluation is used. For the FS calculations, we thought that if a newly developed functionality is similar to the previous one, the effort required for this modification can be assumed negligible. Therefore for the sake of simplicity in our calculations, we assumed “zero” effort for these new functional units. We calculated new functional size based on this condition and recalculated productivity values. The values after similarity consideration can be seen in Table 1. Assuming this large deviation is due to the different characteristics of the applications, development environments or teams; we grouped different application types and performed separate calculations within their dataset. Results are given in Table 2.

As a third question we investigated the effect of FS on different project tasks. HW and ES projects have lower values of similarity. Moreover, duration of the test phases of these applications is not comparable with the duration of GUI type systems.
Therefore to observe how increased percentage of similarity value impact effort in any specific phases we selected only GUI applications. We eliminated one of GUI projects since it has no documentation for phase effort.

Table 2 Average productivity rates and average deviation of productivity

<table>
<thead>
<tr>
<th>App.Type</th>
<th>No Func.similarity</th>
<th>Func. Sim. Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
<td>12,11</td>
<td>6,96</td>
</tr>
<tr>
<td>HW</td>
<td>3,43</td>
<td>0,55</td>
</tr>
<tr>
<td>ES</td>
<td>3,00</td>
<td>2,04</td>
</tr>
</tbody>
</table>

In Figure 1 below, we presented how effort per unit functional size changes with functional similarity percentage of the project. Functional similarities are decided using requirement phases’ results, therefore efforts on man/-day basis of other phases are considered for this observation. For calculation of the unit effort initial functionality size is used, since similarity issue is considered in other axes of the graph.

Fig.1 Effort /cfsu variation w.r.t. functional similarity in different phases

Besides research issues aimed, we reached some valuable information for effort estimation from our data-set. For example, effort for the requirement changes, HW related problems and safety related issues were collected separately during the project lifecycle. By looking at the Table 3 below we can say that for HW and ES Applications for this company, a large amount of effort is required for safety related issues. GUI applications on the other hand, utilize major effort for requirements changes.

Table 3 Percentages of special effort/total project effort.

<table>
<thead>
<tr>
<th>Application Type</th>
<th>%Req Change Effort</th>
<th>% HW Problems</th>
<th>%Safety Related Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
<td>19,6</td>
<td>1,2</td>
<td>0</td>
</tr>
<tr>
<td>HW</td>
<td>0,7</td>
<td>16,07</td>
<td>21</td>
</tr>
<tr>
<td>ES</td>
<td>0,35</td>
<td>1,4</td>
<td>19,1</td>
</tr>
</tbody>
</table>

4. Results and Conclusions

In this study we concentrated on the difficulties of size and effort estimation. In our case study we centered on one of the structure related problems of software sizing namely functional similarity. As it can be seen from Table1 and Table 2, for GUI projects, by considering functional similarity we can achieve a better size and effort correlation. In other words we were able to obtain acceptable average productivity values with this approach.
Variance of the productivity value of GUI projects became reasonable. However for the HW and ES type of applications there is not much change. Their functional similarity percentages are lower. Although for ES applications variance has decreased, we still can not reach an acceptable productivity variance. For these types of applications it seems that the problem cannot be solved with the functional similarity approach we have utilized. We hypothesized that for the effort estimation of these types of applications, size should not be the only driving factor. We observed that; although the number of projects is limited to decide on, HW related projects have comparable number of average productivity values with each other and whenever a new project started, a rate may be agreed on for a better estimation.

Table 3 presents that, safety related effort is found high for ES applications and requirement changes require extra effort for GUI applications before the delivery of the products. Therefore specialized contents of VAFs for specific type of applications could be developed for more accurate effort estimation. Non-functional aspects of software can also be further investigated for agreeable size and effort relationship.

Finally, as it can be seen from the Figure 1 above, we found that for GUI applications, design and coding phases are affected from the similar functionalities of the project. Even for a small number of similarities there exists an effort reduction. However it seems that, for the test and documentation phases, effort per unit of software size does not decrease much. Moreover, we found no relationship between similarity percentage and documentation phase’s effort. Although all application types are affected by similarity, FS is a major concern only for GUI type of applications. One of our constraints in this case study is that we have limited number of projects from HW and ES domains. As a future work, for the validity of our results we are planning to perform root-cause analysis and increase our data-points from HW and ES domains.

Acknowledgement
Special thanks go to Mrs. Derya Arda Ozdamar who, as a Software Analyst and team leader, applied FP estimation method to ES and HW applications.

6. References
Quantified Runtime Performance Analysis of Java Patterns

Dapeng Liu, Shaochun Xu, Monica Brockmeyer
Patterns have been widely used in Object-Oriented design and implementation. However, there is no extensive evaluation specific on their performance. In this paper, we conducted quantified runtime performance analysis by comparing different design solutions to access one value that is specific to the class; in refactoring replacing conditional logic with polymorphism is recommended for this situation. Our experiment demonstrated that although operations related to virtual methods take more time than other statements, the difference of performance efficiency between them gets smaller as transactions scale up. Our concrete experiment data also show the level of Object-Oriented design patterns impact runtime performance, which clearly supports using Object-Oriented design patterns in the performance-critical design.

**Keywords**: Java, polymorphism, design pattern, performance, virtual method.

1. Introduction

As recurring solutions to common problems, design patterns have been commonly used in software design and implementation. However, the performance of using the design patterns in software development has not been extensively evaluated. Although the productivity of the software developer is generally preferred with the expectation that runtime performance can be made up by faster hardware. For those applications in pursuit of ultimate performance or running with limited resources, more attention should be paid to the runtime performance. Moreover, when making a choice among comparable implementation approaches, performance can tilt programmers’ decisions.

In order to evaluate the performance of design patterns, we implemented five different design solutions for a common problem: accessing one value that is specific to the class, and compare their performances. While our experiments demonstrated that legacy procedural implementation is still faster than those with the Object-Oriented programming approach, e.g., normally-used polymorphism approach is at most 4.42 times faster than the legacy if/then branch operations takes; the difference is becoming smaller and more stable (time ratio is 2.11) when transactions become scaling up. This indicates the deficiency can be easily amended by increasing the computing power. We also found that calls crossing class boundary consume more time than inside-class calls, especially when there are not too many transactions, and the maximum ratio of performance is 3.5. Moreover, the time needed to instantiating an object keeps constant if there is enough memory.

Our study can help the developer to have more confidence to take advantage of Object-Oriented features in design, even for performance-critical system. In addition, our study also supports class encapsulation and does not favor redundant and unnecessary wrapping, i.e., to increase inside-class coherence and decrease inter-class coupling.

The rest of paper is organized as follows. Section 2 describes related work. Section 3 describes the experimental settings. Section 4 describes and discusses the experimental results. Section 5 concludes the paper and outlines the future work.

2. Related Work

Design patterns [1], as the recurring solutions to common problems, have attracted a lot of interests and have been widely accepted. After Fowler proposed the refactoring concept [2], patterns have been more appreciated by developers as they could be easier achieved and their benefits were clearly revealed by comparing to previous imperfect design. Patterns and refactoring were proposed to improve the designs constructed with legacy programming logic instead of Object-Oriented.

Raymond [3] pointed out that function calls on Unix were implemented with much more cost than other instructions; however, it was still advised to use functions in system design. In C++, functions can be inlined to improve the runtime performance; even a virtual function may be inlined [4] if appropriate. Generally, modern compilers can do a large amount of optimization to increase the performance of the object code.

To make software flexible, scalable, and distributable, several managed application frameworks have been proposed; one of which is J2EE [5]. In Enterprise JavaBeans 3.0, some new norms that are in favor of coding convenience and runtime performance...
have been proposed. Some other frameworks, such as Hibernate [6], also provide devices for developers to tune up performance.

Although runtime performance has always been one of design focuses, quantified research has not been done widely. Precise performance evaluation can help developers to gain confidence in making decisions in their design. For example, Log4j [7] gave concrete information about its performance as below, “On an AMD Duron clocked at 800Mhz running JDK 1.3.1, ... Actual logging is also quite fast, ranging from 21 microseconds using the SimpleLayout, 37 microseconds using the TTCCLayout”; those data make developer feel confident about recording thousands of events without being worry about possible performance degradation.

3. Experimental Setting

3.1 Benchmark Program

In order to improve the programming productivity, a few patterns have been proposed. In this paper we deal with a scenario that has been widely cited in various pattern tutorials: the class hierarchy has a common method that should be implemented differently by each class [2]. We implemented five different solutions, some of which are accepted by the public as being elegant; some others are against as impeding maintenance.

The class hierarchy is shown in Figure 1. The class structures have enough information that is needed by all the five approaches. Since class fields cannot be overridden, the child classes initialize the field with different values in the class initialization areas. (It is called shadow if one child class declares a field with the same name and type as one field defined in the base class; and the two fields are not related although they look like the same.)

The five implementations are briefly illustrated in Table 1. Due to the limited space, only key parts are shown and all the similar code is eliminated.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Related code (abridged)</th>
</tr>
</thead>
<tbody>
<tr>
<td>instanceof</td>
<td>if ( arr[i] instanceof TestClassChild1 ) { sum += 1; } do the similar operation for child 2&amp;3</td>
</tr>
<tr>
<td>polymorphism</td>
<td>sum += arr[i].getValue();</td>
</tr>
<tr>
<td>type field</td>
<td>if ( arr[i].type==1 ) { // TestClassChild1 sum += 1; } do the similar operation for child 2&amp;3</td>
</tr>
<tr>
<td>by Helper class*</td>
<td>sum += arr[i].getValueByHelper();</td>
</tr>
<tr>
<td>using int array</td>
<td>if ( types[i] == 0 ) { // TestClassChild1 sum += 1; } do the similar operation for child 2&amp;3</td>
</tr>
</tbody>
</table>

* the helper class is illustrated as well which uses one more layer of wrapping that crosses class boundary. Constrained by limited space, the complete class diagram is not illustrated.

One approach, which has been criticized, is to use `instanceof` operator and `if/else` branch clause; with one criticism: the high cost of `instanceof`. A legacy of traditional implementation is to set a flag in each class and then use `if/else`. Basically, branch operations are deemed as inelegant or inefficient in coding. An even worse implementation is to save the types of objects in an outside array; this approach begets serious update problem since a concern distributes in multiple places. The common recommended approach is to add a placeholder in the parent class and override it in each descendant class with desired implementation; in refactoring this is named as “polymorphism instead of branch”. We implemented all these different approaches; meanwhile, we added another solution that uses a helper class to fuse the call to the wanted operations, which is used in other patterns.

In order to compare the five approaches, an array of mixed objects of the three child classes is instantiated by three different approaches: calling newInstance method of the class, using direct new, and calling clone method. For the first two approaches, Java class loader learns about the class of the wanted objects first and then instantiates an object for it; the last approach does involve
extra work: copying the content from the original object to new one.

After creating the array of randomly mixed objects of the three child classes, the test program imposes the five approaches on the array one by one and records the running times. In each iteration, only one approach is chosen to run.

Times used in various implementations to create objects and to get the value specific to each class were record separately. We created a utility Timer for this job, which reads current system time in nanoseconds. To reduce the effect of interference, we record the time for the whole loops, which might be slightly larger than the real times needed to create objects or get the wanted values.

The code was written in Java and compiled with Eclipse 3.1.1 using its Java 5 compiler.

3.2 Experiment Setup

We ran the test program for the Object arrays of different sizes; 5K, 10K, 25K, 50K, 100K, 250K, 1M, and 5M. For each size, we run the program three times and calculate the averages of data. The bench computer has a Core 2 Duo CPU T5270 with 2GB memory and 160GB hard disk. JDK 1.6 was installed on that computer; however, Eclipse 3.1.1 has a Java 5 compatible compiler. Microsoft Excel was used to analyze the data and draw charts.

When we first tried to instantiate 5M objects of one of the three child classes, with default the heap size setup, JVM did not have enough memory to allocate and thus exited with an error. Then we increased the heap size to fixed 300MB (-Xms300M –Xmx300M), all tests were passed successfully. To reduce potential impact of garbage collection, we used a fixed heap size of 600MB (-Xms600M –Xmx600M) in all tests.

4. Experimental Result and Discussion

4.1 Experimental Result

The average transaction time for the five approaches is shown in Figure 2. We are surprised that the blamed instanceof does not cost much more time than virtual calls, which was incorrectly claimed in some technical documents [2]. Figure 2 also shows that accessing to the fields of classes costs more than accessing to primitive types; although Java internally creates an object array for primitive arrays.

![Figure 2 Average transaction time of the five approaches](image1)

![Figure 3 Average transaction time of the five approaches](image2)

For transactions between 25K and 250K, all five average transaction times reduce obviously. For small transaction number, the helper approach consumes much longer in terms of time than any other approaches. Where there are 5K transactions, the helper approach takes 3.5 times as long as the polymorphism approach does. For transactions 5K and 10K, all other approaches except for the helper approach show a comparable performance. When
there are more than 250K transactions, all approaches exhibit comparable performances.

The part of Figure 2 where transactions are larger or equal to 50K is enlarged along the vertical axis and shown in Figure 3, in which illustrations are kept unchanged from Figure 2. We can see that the operation `instanceof` even performs faster than virtual calls. Accessing to primitive array is always constant in terms of time. Normally-used polymorphism approach costs more than double of the time used in the obsolete type array approach. The helper approach always consume the longest time, although the extra caused by it is getting smaller.

In addition to the transaction time, we also observed the time taken to instantiate objects, which is shown in Figure 4. We can see that the time taken by the new and clone operations is relative constant while `class.newInstance` even improves its performance when transactions scale up. When instantiating 5K objects, `class.newInstance` takes 8.44 times as long as that taken by the direct new operation. It is a surprise since theoretically `class.newInstance` uses an implementation similar to that of direct new operation. Clone costs 1.7 times as long as that of the direct new operator since it requires addition operations on the new object.

We also use polymorphism as the benchmark, and plot the ratio of instantiation and method call in Figure 5. From Figure 5, we can find that the object instantiation takes obviously more time than that taken by the virtual method calls.

### 4.2 Discussion
First of all, we omit the possibility of error accumulated by the imprecise timing, since the accumulated error should be on the same significance level as the number of operations; however, the differences of times are three folders of magnitudes.

Although in our case there is only one field, a class that does not have an explicit base class and actually inherits the class Object and therefore owns all fields contained in Object. This hidden fact cannot explain the case that class.newInstance takes obviously longer time than direct new operator does. Anyway, JVM must do extra work when it runs class.newInstance when it is compared to the direct new operator.

Method calls take more time than the sequential statement execution. Accessing to objects needs time; accessing to class fields takes more time than accessing to primitive variables, although JVM actually wrap arrays with objects automatically. Since all methods in Java are virtual, we cannot avoid the time needed by polymorphism. Interactions across class boundaries cost more time than those inside the same object. This observation supports class encapsulation and is against redundant and unnecessary wrapping, i.e., to increase inside-class coherence and decrease inter-class coupling.

4.3 Threats to Validity

The test program is a simplified emulation of real cases. First of all, class structures are very simple and the class hierarchy is shallow; since every method in Java is virtual, the number of methods can affect the time to locate a target method; therefore, virtual method calls may perform worse (According to technical documents, the virtual table is a linear vector). In addition, the depth of the class hierarchy might affect the speed of instanceof. In real case, the performance of instanceof may be remarkably worse than that observed in our experiment. As the helper class runs actions crossing the class boundary, its performance may change to more extent than other approaches.

While there are different flavors of JVMs, how much the impact brought by the JVM is still unknown through this single experiment.

With the consideration that Java is inherently complex, such as supporting multi-threading natively, we do not try to explain our observations, especially with the fact that JVM is running upon another complex OS.

One real program problem may help us understand how complex it can be as related to performance. One standalone Java program ran on a powerful Unix server; in peak time the CPU time could go up to more than 90%. Monitoring CPU load using JProfiler found that most of CPU time was consumed by JAXB operations parsing XML. After switching JAXBContent to be a singleton and avoiding continuously creating new instance of it, the CPU load lowered down to less than 2%. Other system resources, such as IO or network communication, were not considered in our evaluations, systems needing performances in those aspects should be investigated closely case by case.

5. Conclusion and Future Work

Although patterns have been used in Object-Oriented design, there is no much study in evaluating the performance of patterns. In this work, we investigated the runtime performances of various solutions for a common problem by conducting experiments. The result demonstrates that the difference of performance efficiency between using virtual methods and using other statements become smaller when transactions scale up. We also observed that the level of Object-Oriented design patterns have some impact on their runtime performance, which clearly supports using Object-Oriented design patterns in the performance-critical design. In addition the experiment result favors using the class encapsulation and against using unnecessary inter-class communication.

We plan to extend our investigation on some other patterns and adopt various real-life software programs working in different scenarios in our study. In addition, we would like to cover memory consumption. With the consideration of the difference between JVMs, we plan to compare the performances of the same code after compiling and running them on different platforms.

6. Acknowledgements

Shaochun Xu would like to acknowledge the support provided by Laurentian University Research Fund (LURF), Canada and the support provided by the School of Software Engineering, Yunnan University, China.

7. References
